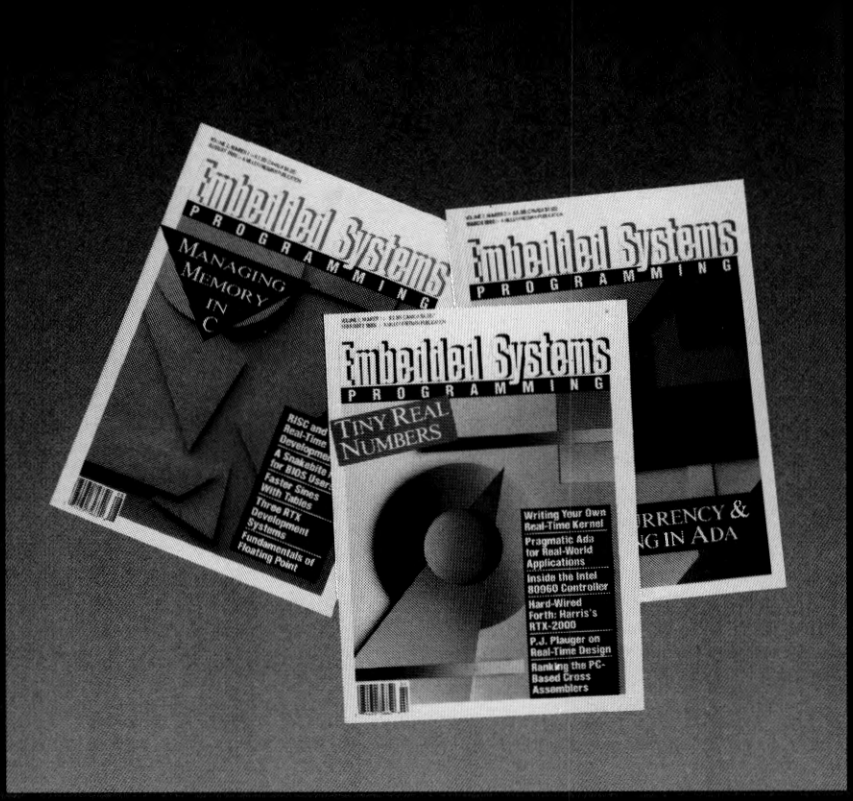
HARRIS SEMICONDUCTOR

Reprint Series



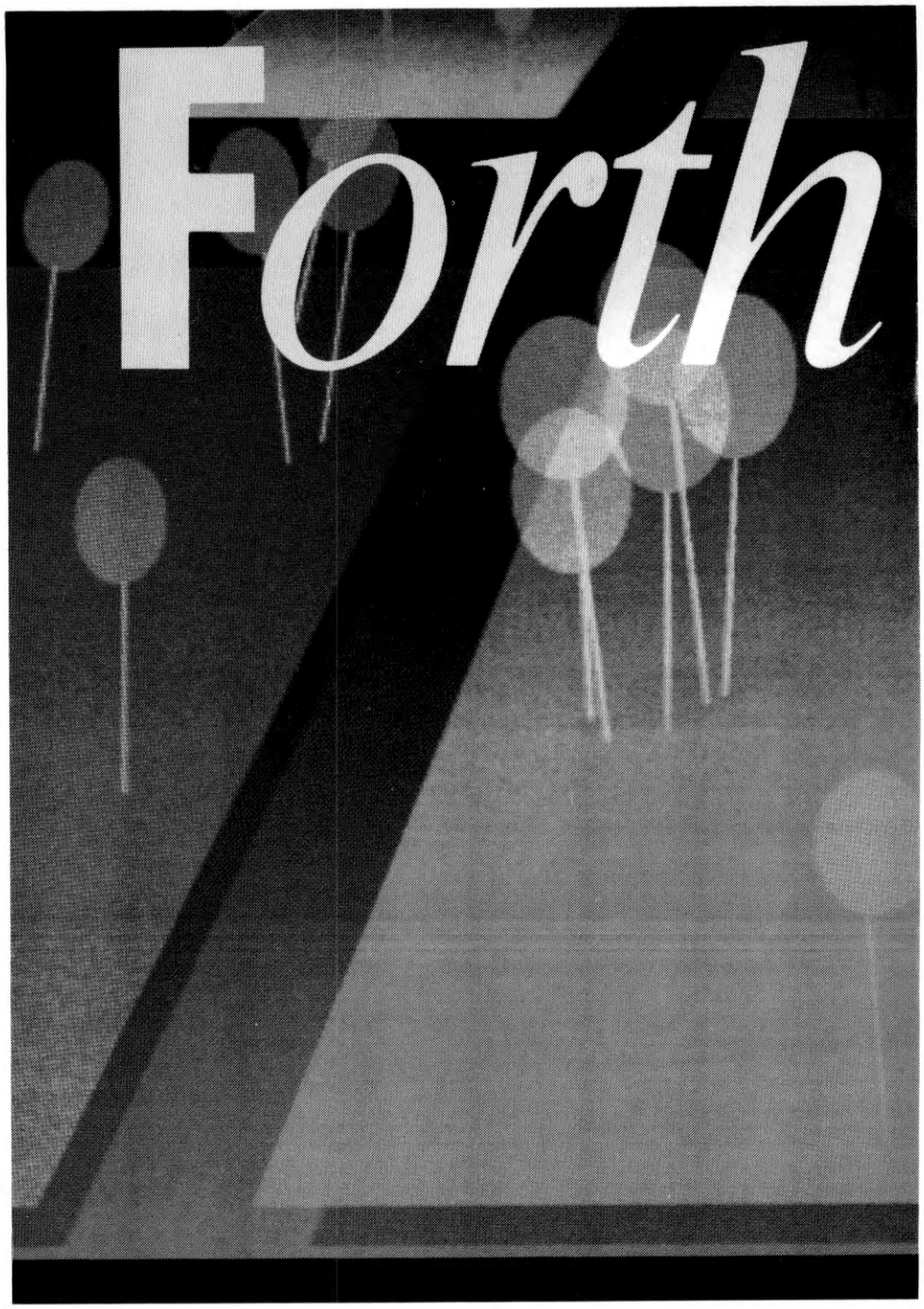
RTX Architecture Benefits and Development Support

TABLE OF CONTENTS

HARRIS

BY ERNEST L. MEYER

Hard-Forth



C is fashionable these days, but for some real-time tasks Forth could be a better choice. In fact, rumor has it that a number of programmers actually write C code during the day and then rewrite the functions in Forth at night. The managers think the code is in C, but the programmers can get the application working much more easily using Forth.

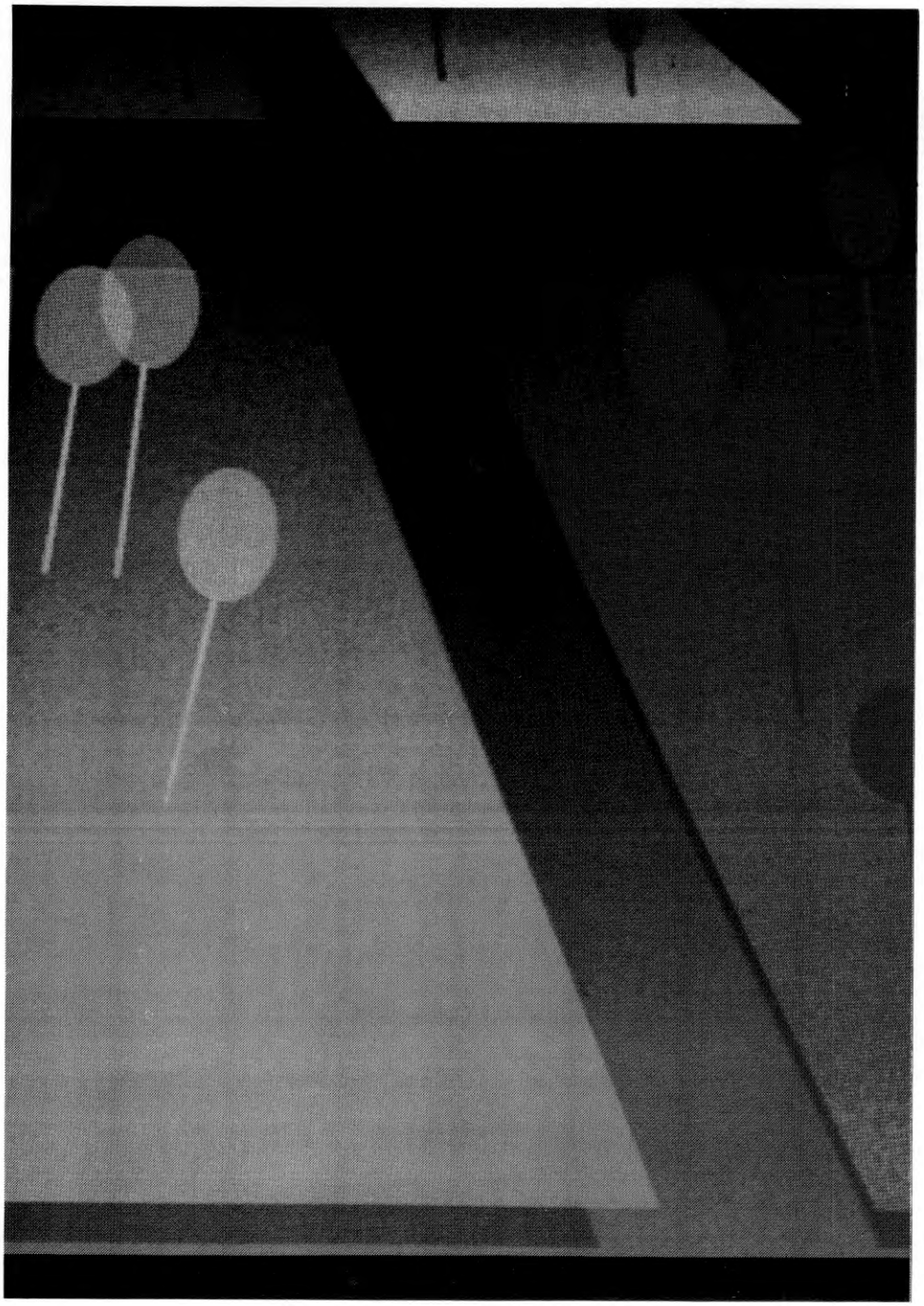
With the advent of Forth-based microcontrollers, programmers no longer need to delude management in such a devious manner. The RTX microcontroller family, now available in production quantities from Harris Semiconductor, uses a subset of Forth instructions as op codes. Because there's no intermediate assembly language between the high-level constructs and the final machine code, the Forth instructions written by the programmer have direct and predictable machine-code equivalences.

Not only does this simplify the construction of efficient real-time routines, but once it's decided that the RTX really is the best choice for an application, the programmer is free to write the application code in Forth.

The processor itself can be customized in hardware for a wide range of specific application requirements. Additional ALUs, multipliers, registers, and stacks can be added right inside the microcontroller's main data paths.

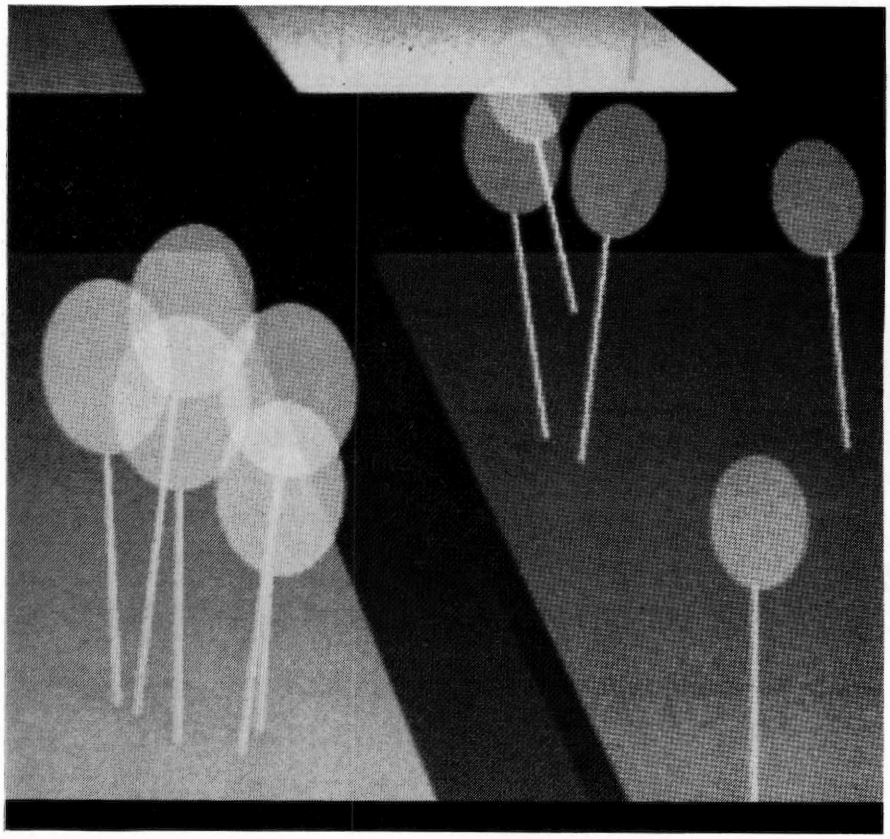
Alternatively, on-chip stacks could be extended or a cache or UART added; D/A and A/D interfaces could even be added on-chip, although the analog circuitry would make the chip somewhat more difficult to manufacture.

Wired



Harris Semiconductor's family of RTX microcontrollers uses a subset of Forth instructions as op codes. There's no intermediate assembly language, so Forth instructions written by the programmer have direct and predictable machine-code equivalences.

Hard-Wired Forth



FORTH-BASED HARDWARE ARCHITECTURE

In the abstract, a chip with a subset of Forth instructions as op codes has a number of advantages for real-time embedded applications. For example, think of the benefits Forth offers for real-time tasks. It's fully reentrant. Direct control of the data and instruction stacks allows more intimate contact with branches, conditional loops, and interrupt functions than other languages usually offer. And reverse Polish notation, because it more closely represents the actual activity of the processor, allows the designer to think more clearly about what's happening than do the "easier" code- and data-ordering mechanisms used by other high-level languages.

On the other hand, no chip will ever reach the ideal, however close it manages to approach it. A chip that uses Forth instructions as op codes is bound to have some drawbacks. Most obviously, it's not the ideal choice for other high-level languages. With a more generic architecture, it's certainly possible to use different high-level languages for different applications and still use the same chip. Although cross-compilers will no doubt become available for the Forth chip at some stage, the chip is unlikely to offer the same level of performance with other high-level languages as it does with Forth.

IRRESISTIBLE FEATURES

At a pragmatic level, the 16-bit RTX microprocessor is not cheap, with a unit cost of $190 in 1,000-piece quantities and a price tag of $3,000 for the complete development system (not including the host computer). And the clock speed, at 10 MHz, is comparatively low.

Other manufacturers offer 16-bit microcontrollers with similar clock speeds at a substantially lower cost. Higher-speed microcontrollers are also available at a debatably lower cost. And Forth compilers can be used for the more generic microprocessor hardware. Even if the compiled Forth code uses twice as many clock cycles on a generic 25-MHz microcontroller as would the same application on the 10-MHz RTX microcontroller, the final system still works faster with a generic processor running at a significantly higher clock speed.

The RTX microcontroller does offer four advantages over generic microcontrollers—predictability, code language uniformity, lower system speed, and potential customization—that can offset the disadvantages.

First, because high-level instructions with direct machine-code correspondences are used, it's much easier for the code developer to determine how long a task will take. This is particularly important in real-time applications, where operations must be completed within strict time slots.

Second, the developer can use the same language for critical functions as for the rest of the code. Since applications tend to use 10% of the code 90% of the time, the critical sections are usually hand-coded in assembler. If the machine code is simply a subset of the high-level code, however, there's no need to

The RTX offers several advantages over generic microcontrollers—predictability, code language uniformity, lower system speed, and potential customization—that can offset the disadvantages of high cost and reduced performance when used with other high-level languages.

link routines in different languages. Writing all the code in Forth also simplifies debugging.

Unfortunately, these advantages mean very little to hardware designers. Luckily, two other benefits offered by the RTX methodology should sway them. First, the use of a 10-MHz rather than 25-MHz microcontroller results in lower system speed, so the other chips in the system don't need to run at ultra-high speeds. And since cache memories aren't necessary at 10 MHz, system cost is lower.

Lower-speed components also simplify board design. Even CMOS becomes difficult to handle at speeds over 20 MHz, when transmission line effects can cause erroneous device triggering and corners in the wire traces can cause signal reflections. The hardware designer will therefore prefer a lower-speed processor from the standpoints of board design effort and overall system cost, even if the processor itself is more expensive.

A second advantage for hardware designers is that the structure of the RTX chip permits them to customize hardware for particular requirements by adding functional units to the microcontroller core. The functions can then be accessed directly by the software as an integral part of the microcontroller.

Figure 1 is a high-level diagram of the chip core. The user can hang additional functional units on the ASIC bus (visible on the right side of the figure). If speed isn't a critical issue, functional units can be hung on the ASIC bus off-

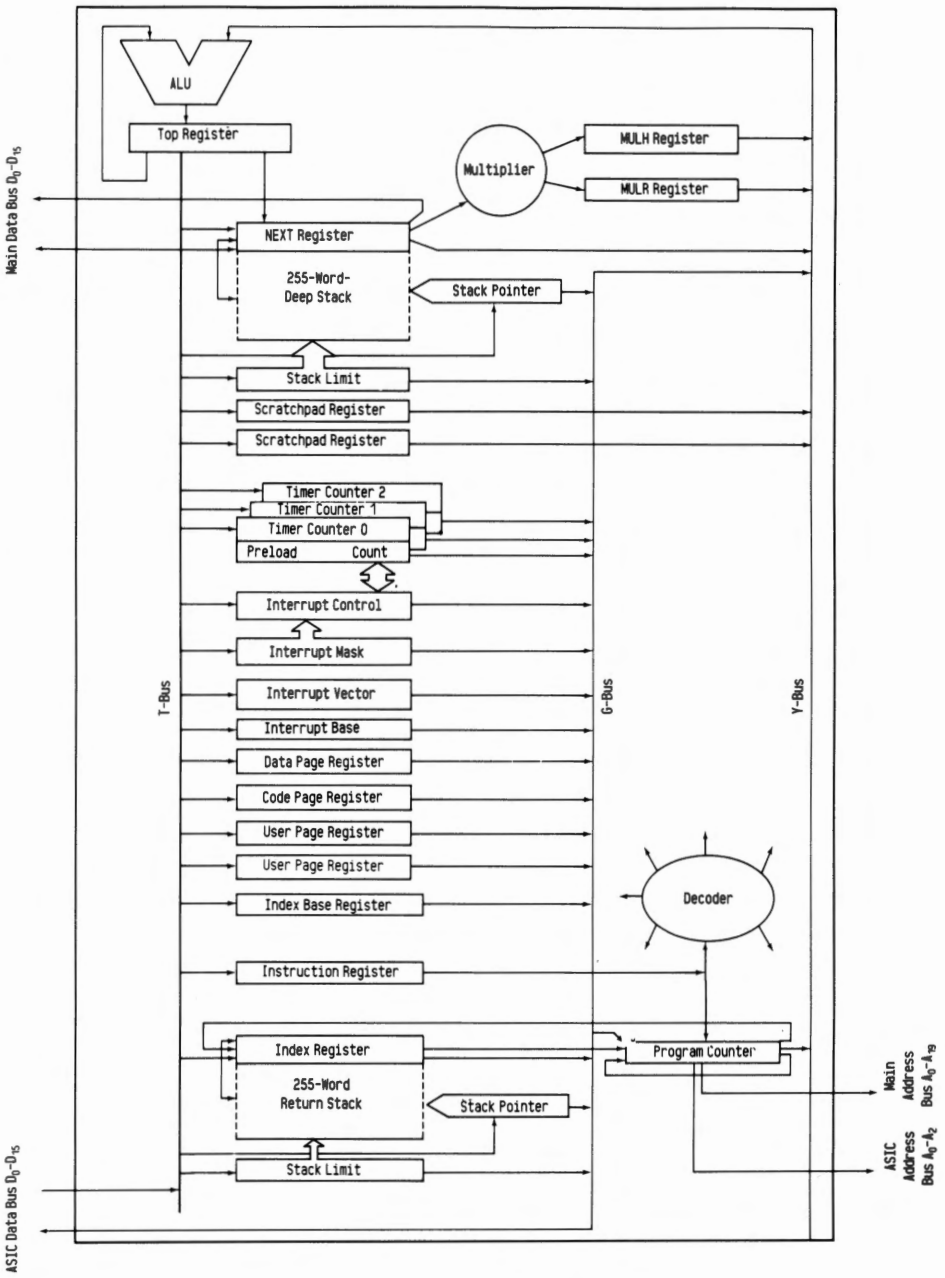


Figure 1
High-level diagram of RTX microcontroller core.

Hard-Wired Forth

chip. Up to eight devices can be directly addressed on the ASIC bus via address lines A0-A2. A full bidirectional 16-bit data bus is then available between the additional functional units on the ASIC bus and the microcontroller core.

If speed is a paramount issue and cost isn't a severe limitation, the additional resources can be placed on-chip. Although this raises chip cost substantially, the added peripheral functionality works about an order of magnitude faster. System cost may also be lower due to the higher degree of functional integration, which decreases the necessary PCB real estate for the system and increases system reliability.

Of course, a similar hardware customization approach is available from other vendors. Two examples are the HPC core from National Semiconductor and the superintegrated products from Zilog. The register-based structure of Forth nevertheless yields a somewhat higher ratio of performance improvement with RTX customization than with most other processor architectures, where added functionality on-chip can't be treated as simply an integral part of the processor itself.

If speed is a paramount issue and cost is not a severe limitation, resources can be placed on-chip. Although this raises chip cost substantially, the added peripheral functionality works about an order of magnitude faster. System cost may also be lower due to the higher degree of functional integration.

WHY IS THE HARDWARE THE WAY IT IS?

Let's take a closer look at the RTX architecture shown in Figure 1. The chip revolves around registers and stacks; there's no cache, and pipelines are minimized.

Instructions that don't go off-chip are either register-to-stack or stack-to-register. All such on-chip operations, with the exception of complete multiplies, take a single clock cycle (100 nsec). Operations that manipulate off-chip data and hardware resources take two clock cycles.

Virtually all the registers, buses, and stack words on the chip are a full 16 bits long. There are two stacks in the chip hardware, both of which are 255 words deep. The stack limit and stack pointer, both eight-bit values that indicate stack status, are available in separate registers. Since the stack limit register is writable, stack depth can be expanded beyond 255 words in off-chip memory. The top two items on the data stack, TOP and NEXT, are addressable as registers. The single top item on the address stack, INDEX, is also addressable as a register.

The ALU, at the top of the figure, always stores its results in the TOP register. When there are consecutive ALU operations, TOP is pushed into NEXT. After any ALU operation, TOP can be transferred to any register except the multiplier. The ALU can perform adds, subtracts, shifts, and Boolean operations; a version of the RTX microcontroller with an enhanced ALU that supports additional single-cycle operations, such as ROTATE, is in the works.

Data is always shifted on- and off-chip through the NEXT register. Not shown in the figure is the SWAP unit, which can swap the first and second bytes in the NEXT register at any time without cycle overhead. As a result, the chip can be used with both "big endian" and "little endian" memory architectures, permitting compatibility with both Motorola- and Intel-like memory-addressing schemes.

Multiplies and divides can be performed by loading the two multiplicands sequentially into the multiplier unit from the NEXT register. The least significant byte of the 32-bit result is readable from the MULH register in the next clock cycle. The upper eight bits of the result can then be read in the subsequent clock cycle by pushing the contents of MULR into TOP. (The old contents of TOP, the eight MSBs of the result, are automatically pushed into NEXT.) Two scratchpad registers are available for storing intermediate values in divide and negative-exponent calculations.

The chip also contains three counters that can function in various modes. The counters can be read independently into TOP and can generate a separate maskable interrupt. Interrupts from each counter shift program control to an interrupt routine pointed to by separate interrupt vectors. The vector for the current interrupt level is readable in the interrupt vector register; it can be changed by writing to that register. Because interrupt vectors are offsets from the address stored in the interrupt base register, counter interrupts can have different effects at different program levels (ensuring a reentrant program).

The five page registers expand the addressable memory space to 1 Mbyte. Four of the 16 nonoverlapping pages are directly addressable at any one time for code, data, user, and interrupt base addresses. A five-bit index base register contains the five MSBs of the index register address, expanding the addressable return stack space to 21 bits.

The instruction address is stored in the program counter. The instruction is then loaded into the instruction register, executed, and decoded. If no branches or loops intervene, the address is incremented and stored in the program counter. Loop counts and branches can be stored in the return stack; when a

Hard-Wired Forth

loop is executing, the index register automatically decrements the remaining loop count. Eight-bit stack limit and stack pointer registers are also available for the return stack.

The architecture of the RTX microcontroller is designed to allow the execution of as many instructions as possible in one clock cycle. By understanding how to use this internal parallelism, you can write highly efficient code.

MAXIMIZING CODE PERFORMANCE

The architecture of the RTX microcontroller is designed to allow the execution of as many instructions as possible in one clock cycle. Unfortunately, it's not always possible to complete all the instructions within the 100-nsec duration of one clock cycle. In fact, there are three kinds of long instructions: off-chip reads and writes, ALU-dependent instructions, and multiply operations. The chip's internal hardware is organized to permit parallel instruction execution in these cases. By understanding how to use the chip's internal parallelism, you can write highly efficient code.

Figure 2 is a state diagram for the chip's instruction execution sequence. Each 100-nsec clock cycle is divided into two 50-nsec machine cycles separated by the rising and falling edges of the clock pulse. Only one off-chip data access (instruction fetch, data read, or data write) can take place in each cycle.

Instruction decoding always takes half a clock cycle, and nothing else can happen at the same time. At the left of the state diagram is one instruction decode operation. If the instruction that's decoded in the first half of the clock cycle doesn't require an off-chip data read or write operation and isn't a multiply operation, it can probably be completed in the second half of the cycle. At the same time, the next instruction can be fetched from off-chip and be ready for decoding in the next clock cycle.

It's clear that the chip does in fact have a two-stage pipeline for instruction

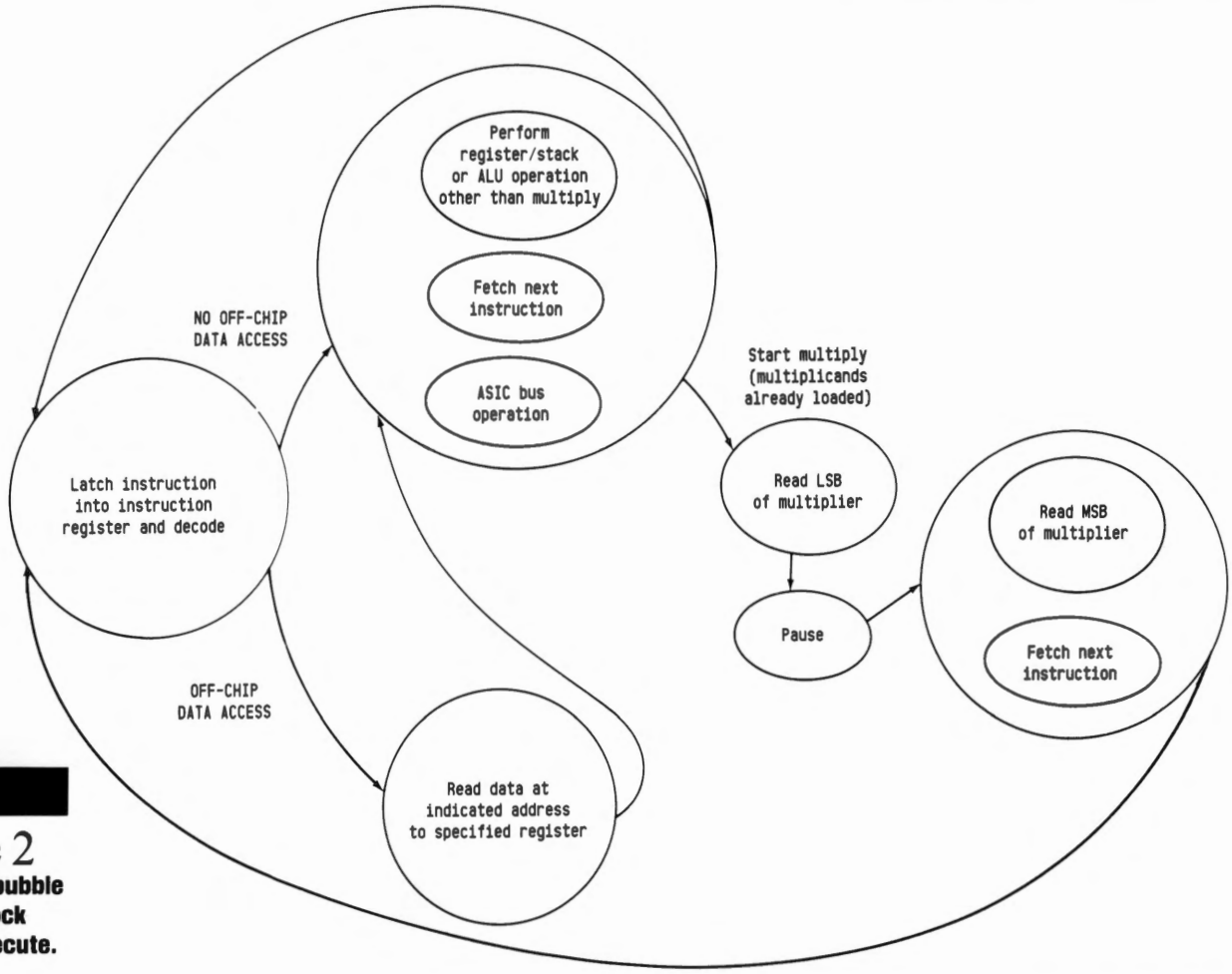


Figure 2
Each state bubble takes ½ clock cycle to execute.

Hard-Wired Forth

prefetch, despite company literature that claims otherwise. Since the pipe is no more than a clock cycle in length, however, the impact of this pipeline isn't as great as that of the seven-stage pipelines found in some of the more complex CISC cores.

ON- AND OFF-CHIP OPERATIONS

Since instructions can be fetched and executed at the same time, at least two parallel operations can be executed in one clock cycle. Indeed, because there are four buses inside the chip, it's possible to have up to four instructions working in parallel. Data can move on all four buses simultaneously.

As mentioned earlier, on-chip operations are either stack-to-register or register-to-stack. Because there are two stacks, a write to one stack can occur in parallel with a read to the other stack. Two off-chip data buses—the main data bus and the ASIC bus—make it possible to move data to or from the two stacks and on- or off-chip, all at the same time.

For example, it's possible to concurrently fetch a data item from a peripheral on the ASIC bus, push it onto the data stack, force a subroutine return (which pops a return address into the program counter), and fetch the next instruction. (These instructions are actually executed in parallel by the Forth word `8 G@ ;`.)

This leads to the first rule of RTX programming: try to alternate use of the two stacks and two data buses to ensure the highest possible level of task concurrency. The direct correspondence of instructions to the actual machine code used by the RTX microcontroller makes it relatively easy to improve task concurrency in the time-critical regions of an application.

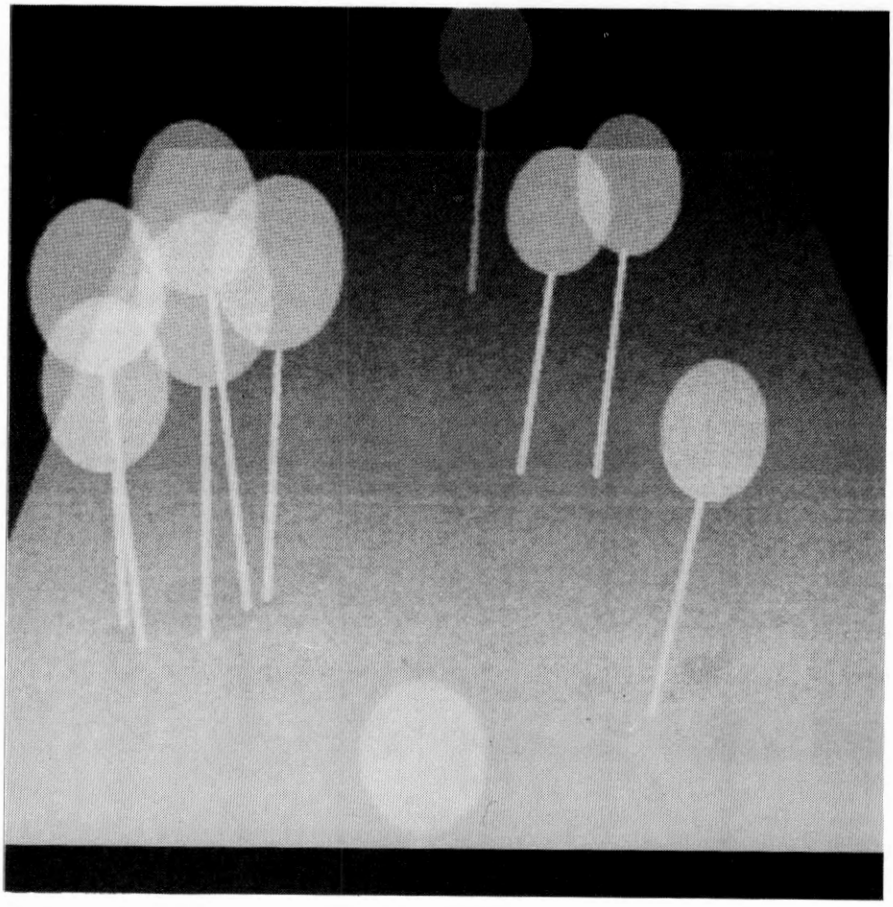
We can see that many internal instructions are highly "parallelizable." There are, however, limitations to the degree of parallelism that can be attained. For example, remember how an instruction fetch and a memory read or write can't occur in the same clock cycle? Because of this, if the instruction involves a memory read or write, then the total execution time will be two clock cycles. However, the next instruction can be fetched during the second clock cycle, during which the fetched data can be processed (as shown in Figure 2).

There are two rules in RTX programming: try to alternate use of the stacks and data buses to ensure the highest possible level of task concurrency, and only fetch data when it's going to be processed immediately by the ALU.

This leads to a second rule: only fetch data when it's going to be processed immediately by the ALU. Just fetching data and flinging it onto the stack for future use wastes hardware parallelism. It's best to keep a static variable out of the stack until it's needed for some arithmetical manipulation by the next immediate instruction. That way, use of the instruction prefetch pipeline is maximized.

MULTIPLIES

Multiplies are special cases that don't obey these generic rules. The company literature states that the chip takes one clock cycle to perform a multiply. This is true, but additional clock cycles are needed



Hard-Wired Forth

to load the multiplier and read the results.

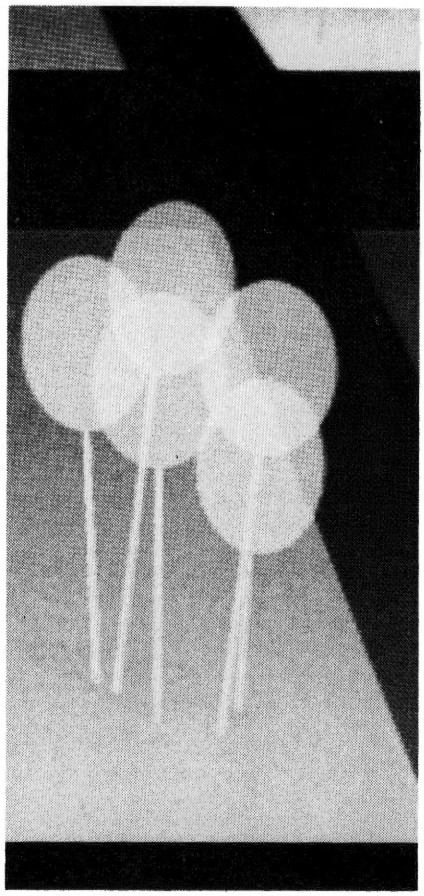
To perform a multiply, the programmer loads the two multiplicands into the MULH and MULR registers from the NEXT register. The eight LSBs of the result can then be transferred into TOP in the next cycle by the instruction READML. As noted earlier, the eight LSBs are pushed into NEXT and the eight MSBs are pushed into TOP in the following clock cycle. This can be accomplished using the instruction READMH.

A full multiply therefore takes four clock cycles if the multiplicands are already resident on-chip. If one multiplicand is fetched from off-chip, a full multiply takes five clock cycles. If both multiplicands are fetched from off-chip, a full multiply takes six clock cycles. If one of the multiplicands is already resident in the multiplier registers and the other multiplicand is resident in a scratch register or stack on-chip, the multiply can be completed in three clock cycles.

This disparity in multiplier performance between operations for on-chip and off-chip data is especially important when using a data item from main memory as one multiplicand and on-chip data as the other. It makes sense to load the on-chip multiplicand into the multiplier first. The multiply operation can then be carried out in the next clock cycle rather than waiting for the next instruction to load the multiplier with the on-chip multiplicand (a practice that degrades on-chip parallelism).

CODE DEVELOPMENT SUPPORT

Harris supplies a development system for the RTX called the RTXDS, which runs on an XT, AT, or compatible and a development board. The system has a real-time symbolic monitor for debugging, a disassembler, a DOS file interface utility, a Forth-83 compiler for the 8086/286, and an RTX cross-compiler. Code is written in the 80xx environment and cross-compiled onto the RTX board.



Harris' development system for the RTX includes a real-time symbolic monitor for debugging, a disassembler, a DOS file interface utility, a Forth-83 compiler for the 8086/286, and an RTX cross-compiler.

Code development is enhanced by the availability of function libraries. The development system includes a library of words for basic I/O manipulation; a library of Forth words for complex arithmetic functions, including transcendentals, should be available by press time. Harris is also preparing a library of advanced functions, such as vector and Fourier transforms, that should be available over the next year.

Debugging Forth code using the RTX processor differs from using a generic processor in that the RTX contains no trace or single-step utilities. However, since all the internal registers are static, you can stop program execution at any time and examine the contents of the registers by setting a breakpoint with the RTX monitor. During debugging, the monitor can be used for examining and changing target memory locations, registers, and I/O ports. Forth names are automatically inserted by the monitor in place of numeric memory locations.

The development board features 32 kbytes of high-speed static RAM that can be used with the chip for real-time code execution. System control resides in 16 kbytes of EPROM; there are two sockets for 16-kbyte EPROMs as well as a breadboarding area and facilities for adding I/O ports for test purposes. This permits the addition of physical memory maps, wait state generation circuitry, and I/O functions that fully emulate the target hardware.

USING THE RTX MICROCONTROLLER

RTX microcontrollers are being used for medical image scanning, to control robotic arms in cannery lines, and in aerospace applications. In fact, they can perform virtually any high-performance real-time task. RTX microcontrollers are particularly effective for tasks that have a high degree of context switching and where predictability is a foremost issue.

Listing 1 illustrates the use of the RTX microcontroller for real-time tasks. The code allows the RTX to function as a UART without any additional hardware and takes up less than 1% of the available microcontroller run time.

Hard-Wired Forth

It treats one of the ASIC bus locations as a UART running at a user-settable speed. This example illustrates the power of the RTX microcontroller without added hardware. (Although the code does use up all the on-chip timer resources, it can be restructured to use only one timer.)

Full UART functionality can also be achieved by adding a UART onto the ASIC bus. The hardware UART can then be run with the minute code portion shown in Listing 2. As you can see, the necessary code is minimal.

Listing 1

A UART function can be embedded in software while using only 1% of the processor's available time.

```
\Set timer 1 interrupt interval.
   :SETBRG
      VALUE TIMER 1 G!
      16 VALUE * 1BIT !
\************************
\UART RECEPTION
\************************
\Sample incoming data 16 times per bit.
   :TIMER_1_INT
      UART G@
      0001 AND IF NODATA ELSE CHECK_START_BIT
      THEN
\tells us there's no start bit
   :NODATA
      0 2NDTIME !
\Wait before interrupting again.
   :WAIT1
      1BIT TIMER2 G!
   :CHECK_START_BIT
      2NDTIME @ IF GETCHAR ELSE
\There's a start bit—enable interrupt.
\set a flag, and check 1/2 cycle later
\to recheck start bit.
         FFFF 2NDTIME G!
         1BIT 2/TIMER1 G!
      THEN
\Fetch character from UART register and
\shift left: merge data into receive register.
   :GETCHAR
      UART G@
      COUNT @
      0F( 2*
      RXCHAR @ OR RXCHAR !
      COUNT @ 1 +
      DUP COUNT !
      8- IF ENDCHAR
      ELSE WAIT2
\Set timer 2 interrupt interval.
   :WAIT2
      1BIT TIMER2 G@
\Finish up the character.
\Disable interrupt until new character
\is detected.
```

```
   :ENDCHAR
      0 COUNT 1
      FFDF INTMASK G!
\************************
\UART TRANSMISSION
\************************
\Prepare a word for transmission
\by adding stop bits. then check CTS bit.
   :TRANSWORD
      2 * 1 + 2 * 1 * +
      TWORD !
      UART G@
      0040 AND IF NOT CLEAR ELSE TRANSMIT
      THEN
\Transmit a word from the top of the stack.
   :TRANSMIT
      0 UART G!
      FFFF INTMASK G!
      1BIT WAIT3
      0 TCOUNT !
\transmitting individual characters
   :TRANSMIT-INT
      TWORD @ DUP
      UART G!
      2 * TWORD !
      TCOUNT @ 1 + 9 =
      IF DISABLE ELSE WAIT3
```

Listing 2

Running a discrete UART in on-chip or off-chip hardware takes far less code.

```
\Initialize and clear data port.
   :INIT
      PARAM1 UCMD G!
      PARAM2 UCMD G!
      UDATA G(@ DROP
\Read status port.
   :(aSTATUS (--n)
      UCMD G(A
\Poll for and read character.
   :RECEIVE (--n)
      BEGIN (aSTATUS RRDY AND UNTIL
         UDATA G(a 007F AND
\Transmit when ready.
   :TRANSMIT
      BEGIN (aSTATUS TXRDY AND UNTIL
         UDATA G!
```

However, since the software-based UART uses only 1% of the processor's total run time and the code is fully reentrant, it may be less sensible to embed the UART in hardware. And the software UART only uses about 200 bytes of EPROM space.

Ernest Meyer, an independent technical writer and former editor of VLSI Systems Design, *is also a contributing editor and columnist for* Embedded Systems Programming.

by Ray Duncan

Reprinted from Embedded Systems Programming, March 1989. Miller Freeman Publications.

A New Breed of Microcontroller

If you're one of the lucky programmers who received the premiere issue of *Embedded Systems Programming*, you could hardly have missed the four-page color advertisement by Harris Semiconductor for something called the Real Time Express family of microcontrollers. This advertisement made some impressive claims for speed, interrupt response times, and power consumption of the RTX-2000, in particular, but gave few clues—other than the enigmatic phrase "innovative dual-stack architecture"—as to how such impressive numbers were achieved or what the RTX-2000's relationship to other, more familiar microcontrollers might be.

The RTX-2000 bears little resemblance to any processor you're likely to be familiar with (unless you're grizzled enough to have worked on Burroughs mainframes). It's a member of that rare species: the stack machine. Regardless of your current affiliation with one processor school of thought or another (RISC or CISC, Intel or Motorola, and so on), the Harris chip is worth a closer look. It represents a radical divergence from the conventional wisdom on how to wring more computing power and shorter development cycles out of the hardware/software duo.

Of course, almost any CPU will support a stack these days, but a conventional CPU's only real need for a stack is to save return addresses and register contents during execution of a subroutine or interrupt handler. The stack typically doesn't participate in arithmetic/logical operations; operands for such instructions are taken from registers or memory, and the results are returned to registers or memory. Although some high-level languages, such as C, also use the stack for parameter passing, this is only a convention; stack usage may vary drastically from one compiler to another.

What's unusual about the Harris chip? For starters, the RTX-2000 is not a classic Von Newmann machine. Another novel aspect is its use of a sort of horizontal microcoding scheme.

In a true stack machine, the stack is the center of the action rather than an auxiliary area of storage. Operands for arithmetic/logical operations are always taken from the stack, and results are returned to the stack. (If you've ever programmed an Intel 80x87 numeric coprocessor or used an HP calculator, you're already familiar with this concept.) Such machines typically have few or no registers in the traditional sense—that is, registers used as accumulators or indexes—although special-purpose registers to control I/O and to point to the base of the stack and the current instruction are still needed.

The RTX-2000 actually carries the stack-machine concept a bit further by supporting two hardware stacks, the parameter and the return. The parameter stack is used for the operands and results of arithmetic/logical operations, as already described. It's also the source or destination of any instruction that accesses main memory. That is, a value in main memory must be loaded onto the parameter stack before it can be manipulated, and a value can only be written into main memory from the top of the parameter stack. The return stack holds loop counters and return addresses during execution of subroutines and occasionally provides temporary storage for other data.

What else is unusual about the Harris chip? For starters, the RTX-2000 is not a classic Von Neumann machine. It has three distinct address spaces: main memory, which can hold either code or data, and the two stacks, each of which is 256 words deep and has its own address and data buses. The stacks can't be positioned at arbitrary addresses in main memory as they can in conventional CPUs, and arbitrary stack locations can't be accessed with memory fetch and store instructions. This idiosyncrasy has interesting implications that you can appreciate only after you begin to code for such a CPU and discover that your faithful old programming idioms (or clichés) no longer work!

Another novel aspect of the RTX-2000, at least in the microcontroller world, is its use of a sort of horizontal microcoding scheme. The chip has five internal buses that carry data between the tops of the two stacks, the instruction pointer, the ALU, the multiplier, the shifter, and so on. These buses are controlled by separate bit fields in the op codes for arithmetic/logical instructions. Consequently, it's often possible to merge two operations into the same instruction and thus into the same machine cycle. For example, you can duplicate the number on top of the parameter stack and then divide the new top of the stack by two.

This horizontal microcoding also allows the Harris chip to have a one-cycle overhead for invocation of a subroutine. This might sound like preposterous marketing hype, but it turns out to be on the level. The call to a subroutine is, of course, a distinct machine instruction and requires a complete machine cycle, but the return from the subroutine comes for free. How is this possible? All machine instructions—other than the `call` instruction itself—contain a reserved "return" bit field. If the bit is clear, the instruction pointer is incremented during execution of the op code in the normal manner. If the bit is set, the return stack is popped into the instruction pointer in parallel with whatever else the instruction is supposed to be doing, at no additional cost in execution time. Thus, the return from any subroutine can simply be folded into the last instruction of that subroutine.

By this point, those of you who know something about Forth are probably thinking that this description of the Harris chip has a familiar ring to it. That's not too surprising, since the lineage of the RTX-2000 can be traced directly to Charles Moore, who invented the Forth programming language in the late 1960s. Moore was one of the founders of FORTH Inc., the flagship software house for Forth development tools and applications. As Forth matured and stabilized, however, his interests turned to hardware and he became increasingly dissatisfied with the mismatch between the Forth virtual machine and the architecture of conventional CPUs.

In 1980, Moore left FORTH Inc. and began to devote most of his time to experimentation with dual-stack machines. His efforts led to the formation in 1984 of a Silicon Valley start-up called Novix Inc. (funded by Sysorex International) and the design of a new processor called the NC4000 by Moore and Bob Murphy. The first working NC4000 chips, fabricated by Mostek using a three-micron HCMOS process and packaged in a 128-pin grid array, were delivered to Novix in March 1985. The chips turned out to be buggy but usable, which was fortunate since Novix was undercapitalized and overextended almost from the outset and never revised the NC4000 mask to eliminate some severe problems with interrupt handling and step-wise multiplication. Novix did manage to sell over a thousand chips before it faded into obscurity.

The most amazing part of this whole history is the open-mindedness of Harris Semiconductor while most of its competitors rush headlong into RISC technology.

By a strange quirk of fate, the Novix chip caught the attention of some engineers in the CAD/CAM division of Harris, who hoped to use it as a high-speed number-crunching coprocessor in graphics applications. After looking more closely at the chip's capabilities, Harris decided to license the NC4000 design, fix the bugs, add some new capabilities, and incorporate the CPU into its standard cell library. Thus, a modified form of the Novix CPU, together with some timers, on-board high-speed stack memory, and a 16-by-16 multiplier, constitutes the heart of the Harris RTX-2000 microcontroller as we know it today. In retrospect, the most amazing part of this whole history is the remarkable open-mindedness of Harris Semiconductor when most of its competitors are rushing headlong into RISC technology.

The Forth ancestry of the RTX-2000 doesn't mean that it can only be programmed in Forth, however; quite the contrary. Recursive descent compilers for just about any modern, block-structured language generate Forth-like, postfix intermediate code at some point during translation from source code to object code, making the chip an excellent target for such compilers. Harris is beta testing a C compiler for the RTX-2000 that should be officially released by the time you read this; a PROLOG compiler is also under development. In fact, it seems safe to predict that the only kind of translator you'll never find offered for the Harris chip is an assembler. That's because Forth is, in a sense, the RTX-2000's assembler language—all of the chip's machine instructions map directly onto Forth language elements.

Ray Duncan has written columns for Dr. Dobb's Journal, Softalk/PC, *and* PC Magazine. *He's the author of* Advanced MS-DOS Programming *(Redmond, Wash.: Microsoft Press, 1986) and* Advanced OS/2 *(Microsoft Press, 1989). He owns Laboratory Microsystems Inc. (Marina del Rey, Calif.), a software house specializing in Forth interpreters and compilers.*

by Tyler Sperry

Three RTX Development Systems

When Harris Semiconductor launched its "Real-Time Express" campaign last year and unveiled the RTX 2000 processor, it was greeted with mixed reactions from developers. Almost everyone was intrigued by Harris's claim that it had the world's fastest 16-bit microcontroller; an average of 15 MIPS from a 10-MHz clock isn't easily ignored. Those unfamiliar with the chip's genesis, though, were a trifle confused by the dual-stack architecture and the chip's "near-RISC" description. Developers in the Forth community, on the other hand, were much more enthusiastic, as it was clear the processor had been specifically designed to run Forth. But for both groups, the response immediately following the introduction was simple: "Sure it's fast, but what development tools are available?"

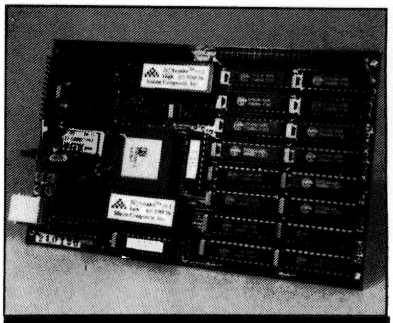
Almost everyone was intrigued by Harris's claim that it had the world's fastest 16-bit microcontroller. But then they said, "Sure it's fast, but what development tools are available?"

A few months back I asked that same question and quickly wound up evaluating three different development systems for the RTX 2000: the SC/FOX SBC from Silicon Composers, the Harris RTXDS, and the Innovative Integration FB2000. The review scenario was the same for all the systems; the goal was to develop an RTX-based solution to a hardware design involving real-time constraints. To even things out, all the systems were to use an IBM PC as the host and communicate via an RS-232 serial link. While all three systems demonstrated admirable hardware performance, their design rationales and software support varied substantially.

THE RTX BASICS

It's no surprise that all these development systems use the stock RTX 2000 processor. The RTX 2000 has been discussed before in this magazine (see "Hard-Wired Forth," Feb. 1989, pp. 58-71, and "A New Breed of Microcontroller," When in ROM, Mar. 1989, pp. 15-18), so I won't go into the processor's design here; suffice to say that the RTX 2000 is a 16-bit RISClike processor designed to implement Forth primitives in its instruction set. Almost all instructions execute in a single cycle, and the processor doesn't have a cache. All the systems I examined were available with a clock rate of either 8 or 10 MHz; all the processor boards used high-speed CMOS static RAMs and EPROMs.

One RISClike aspect of the RTX processor is its lack of microcode: all the bits of an instruction actually toggle transistors and perform operations. Unlike most RISC processors, however, the RTX has several internal buses that connect the various registers to the two internal stacks and the outside world. The multiple-bus arrangement actually allows the programmer (or, more precisely, the compiler) to overlap several instructions into one word. All three of the development systems I examined support this optimization of code.



Available From: Silicon Composers Inc., 210 California Ave., Ste. K, Palo Alto, Calif. 94306, (415) 322-8763
Price: $1,495 (includes software development system)
Support: Free telephone support, 90-day warranty, contract services
System Requirements: Serial cable, terminal, five-volt power supply

FOR INFORMATION CIRCLE #101

SC/FOX SBC

Silicon Composers was a solid supporter of the Novix Forth processor (the chip that served as the original model for the RTX development team), so it's no surprise that it also supports the Harris chip. The company is probably best known for its coprocessor PC cards, but it also provides stand-alone development cards. Third-party Forth packages are available for all the cards from Forth Inc. (Manhattan Beach, Calif.) and Laboratory Microsystems Inc. (Los Angeles, Calif.).

The SBC is a small (100 mm by 160 mm) board that contains the RTX processor, EPROMs, CMOS static RAM, expansion connectors, and miscellaneous "glue" parts. The board is available in a number of memory configurations ranging from 64 kbytes to 512 kbytes, and the bus connectors give you

full access to the ASIC bus for hardware expansion. If judged solely on clean design and hardware functionality, this system would be the winner. Unfortunately, there's a lot more to a development system than just the hardware.

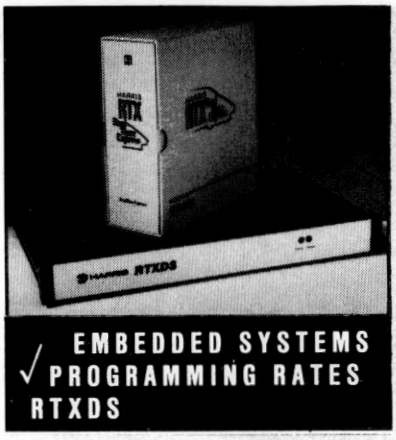
ON THE SOFT SIDE

While the software is adequate for developing applications, it's Spartan in comparison with the alternatives. It took up about 240 kbytes on my hard disk and consisted of the compiler, the loader, and a few libraries and utilities. Installing the software was a simple operation.

The software development cycle on the SC/FOX SBC was a shock—I was used to the seductive, interactive nature of Forth systems running directly on the target system and using the PC strictly for I/O. Alas, the SBC has more in common with traditional C cross-compiling than with traditional Forth development. You edit your source files with your favorite editor; the use of straight ASCII text files and the C-like extension of `#include` for library usage was a nice alternative to the traditional block files. Once you've edited the source code, however, you have to compile it and load it into the target board's RAM. Does this process sound familiar? Does it sound slow? Correct on both counts.

Some explanation of the SBC philosophy came when I contacted Silicon Composers for help. (I'd been converting some old code and had discovered that the Forth compiler didn't understand words like `PICK` and `CASE`.) George Nicol, the company's president, explained that while Silicon Composers had a resident Forth working with its coprocessor boards, the stand-alone board's compiler was designed more as an optimizing assembler/compiler for the RTX processor than as a Forth development system.

The debugging assistance is minimal and not particularly interactive. One of the library files contains a few traditional display words like `.s` and `.wdump`, joined by the more sophisticated `.break`, which performs a breakpoint. At least it's better than sprinkling `print` statements throughout your code.



Available From: Harris Corp., P.O. Box 883, Melbourne, Fla. 32902-0883, (407) 724-7302
Price: $2,995
Support: Free telephone support, 60-day warranty, bulletin board service, training courses
System Requirements: IBM PC, DOS v. 2.1 or later, serial port

FOR INFORMATION CIRCLE #102

A FEW MISSING DETAILS

I found the SBC users manual to be good, with only a couple of annoying problems. I discovered the worst of them indirectly when I tried to use the parallel printer port. The folks at Silicon Composers told me I could get an adapter cable for the board from a local software/hardware emporium. Unfortunately, the salespeople were unaware of their support role for Silicon Composers and didn't have the appropriate cable in stock. I managed to manufacture my own cable without a great deal of effort—or a great deal of success. Another call to Silicon Composers straightened out the problem (the pinout diagram reflected only the pins for the printer end of the cable, not the header-pin assignments on the SBC) but left me with a lingering irritation with companies that don't feel they need to supply schematics or cables for their boards.

The SC/FOX is unique among the systems I tested in two important ways. First, no schematics are available for the board, implying (to me, at least) that the company would like you to develop applications and then buy your hardware from them. This is certainly no crime, but it also doesn't make the SC/FOX SBC my first choice as a development platform where custom hardware would be involved.

The second unique aspect is that the SC/FOX SBC was clearly designed for applications requiring more than the usual amount of RAM. To the credit of Dr. Nicol and crew, they've made it easy for customers to upgrade the board by either swapping boards or installing bigger RAM chips. That's good from the user's perspective, but if I were developing applications that used a lot of RAM (and had a budget to match) I'd be tempted to opt for development using Silicon Composers' coprocessor boards.

HARRIS RTXDS

Anxiety was my initial reaction to the arrival of the Harris development system: the shipping box measured 12 inches by 24 inches by six inches and weighed enough to remind me of the massive crates of documentation that came with Microsoft's infamous OS/2 SDK. Happily, this package is nothing like the OS/2 SDK—in almost every respect, it's a pleasure to work with.

The RTXDS I received consisted of the RTXDB development board, a Harris binder filled to overflowing with documentation, a modified version of Laboratory Microsystems' PC/Forth, and a power supply. The RTXDB itself was clearly devised with the hardware designer in mind: the roughly nine-by-14-inch board contains a ZIF socket for the processor, plenty of space between components, lots of test points, and two separate areas for breadboarding (one area is specifically designed for memory circuits). There's little doubt that this board combined with the Harris hardware documentation is a great hardware development package. As I said before, though, there's more to a development system than just hardware.

The software part of the package worked well, with one noticeable exception: the installation. The PC/Forth in-

stallation process for the RTXDS was conceptually very simple. All the installation batch file had to do was load a binary overlay for the appropriate video display, load the RTXDS overlay, and save the resulting images to disk. In practice, the automatic installation blew up several times. I don't know if the blame lies with Laboratory Microsystems or with Harris; I do know that it's inexcusable for an installation batch file to be shipped without first being checked.

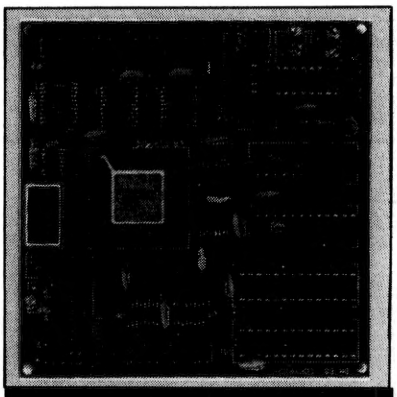
I'm not a Forth guru, but I was able to manually install the system rather quickly. This is due in no small part to the excellent PC/Forth documentation; every time I consulted the manual, I was able to find the information I needed quickly and easily.

I'd rate the documentation from Harris as good. There were a few times when I wasn't able to find things as easily as I should have, and there were times when I found contradictory remarks about the software. Harris has promised to work on the documentation and produce a separate, bound programmer's reference in the next revision (due out sometime this summer).

Once properly installed, the software worked very well. Like the Silicon Composers product, the PC/Forth package doesn't implement Forth running on the target system but instead compiles code on the PC and then loads it into the target's RAM. That's about the extent of the similarities between the two packages, though. Consider some of the advantages of the PC/Forth RTXDS package:

- Forth-83 compatibility.
- A dual-window programming environment that emulates (to a degree) a target-resident Forth system.
- A true RTX emulator for developing I/O-independent code without using the target system.
- Debugging help that includes a memory-access monitor for tracing variable assignments, as well as interactive disassembly and symbol table access.
- Sample code including UART and interrupt handlers.

The combined files of the PC/Forth RTXDS software package took up 520 kbytes on my hard disk, not a bad balance between size and utility.



Available From: Innovative Integration, 4086 Little Hollow Pl., Moorpark, Calif. 93021, (805) 529-7570
Price: $995 (or $1,500 for system with full source code)
Support: Free telephone support for 90 days, one-year warranty on board, one year free software updates
System Requirements: IBM PC with serial port (color graphics adapter desirable)

FOR INFORMATION CIRCLE #103

INNOVATIVE INTEGRATION FB2000

In many ways the FB2000 is both the easiest and the hardest development system to review. If I could do a decent imitation of Bill Machrone, I'd give it an Editor's Choice award. As it stands, I'll have to settle for congratulating Jim Henderson and the folks at Innovative Integration for winning our unofficial Bang-for-the-Buck award. At $1,075 for a developers package that includes a complete polyFORTH system and an EPROM programmer, the FB2000 is a bargain.

For people outside the field of embedded systems, there might not be anything remarkable in the appearance of the FB2000. If you've grown used to development systems the size of a home stereo, though, it can come as a pleasant surprise to pick up a little card (4.2 inches square) and consider its power. The small size of the FB2000—let alone the accompanying EPROM burner, which is even smaller—is a nice hardware demonstration of one of Chuck Moore's guiding principles: small is beautiful.

Apart from the form factor, though, there's nothing small about this package. For example, the various source, documentation, and executable files take up over 2 Mbytes on my hard disk. Libraries and routines for such features as graphics, multitasking, and floating-point support are casually included in the source code as though they're no big deal. (By way of contrast, Silicon Composers sells a four-function IEEE floating-point RTX package for a mere $500.) I hate to be repetitive, but have I mentioned the word *bargain* yet?

In contrast to the SC/FOX SBC and the RTXDS, the FB2000 is a resident Forth package. That is, the PC is used for terminal and disk I/O via the serial link and the developer interacts with a completely functional polyFORTH system running on the target. While there's some delay in loading files via serial link, it's nowhere near as painful as it is with the Silicon Composers SBC. That probably has something to do with the fact that the FB2000 system cranks up the serial rate to 38 kbps or faster.

Despite all the talk about clock rates and wait states from the hardware labs, the measure of our productivity on the software side most often relies on the quality and speed of our programming tools. In that regard, Forth programmers have traditionally had an advantage in embedded systems development; the small size of most Forth systems has allowed developers to do their work quickly and interactively on the target system instead of wading through the delays of cross-development. If ever there were a processor where that advantage was worth serious consideration, it would be the RTX 2000.

LOOKING FOR FAULTS

I tried to find some, really I did, but there weren't that many faults. Perhaps the worst thing you could say about the

FB2000 is that it's the least stable of the systems mentioned here. (It's ironic that the system with the most changes was the system I had the least problems with.) No sooner did I get a system for review than Jim Henderson was on the phone uploading a new READ.ME file with improvements to the port I/O.

The system I got for review had a wire-wrapped EPROM programmer—an improvement added to the developers package because the people at Innovative Integration didn't think their customers wanted to shell out $1,000 for a premium ROM burner compatible with the speedy Cypress EPROMs they use. I called Innovative Integration's offices a month or two later to confirm a few details and it turns out that, yes, the EPROM burner now exists as a finished board. By the way, they're speeding up the disk-file access by adding in-line compression and decompression.

The two documentation manuals are among the best I've seen in Forth systems, so there's no problem there (unless you count all the READ.ME files from upgrades).

One potential shortcoming of the board is the limitation of 64 kbytes of static RAM, but expansion connectors (and schematics) would enable an engineer friend to add another board if you had an application that needed the extra memory.

The biggest short-term problem for developers is the fact that Innovative Integration is, like Silicon Composers, a small company. If you call with a support problem, there's a good chance you'll have to leave a message and wait a couple of hours for a reply. As with the best companies, though, when you get a response you wind up talking with someone who knows the product very well.

THE BUCK STOPS HERE

All these systems have an RTX under the hood, so to speak, so they're all screamers compared to the Forth controllers most of us have seen before. And while there were occasional problems, these are well-engineered products. So how do you choose?

As hinted at in the "Bang-for-the-Buck" comment, the Innovative Integration product is my pick as the best investment for developers interested in investigating the RTX processor. You get excellent performance and value and huge amounts of source code for your applications.

If you have friends in hardware who are seriously considering building their own boards based on the RTX 2000, however, I don't see a real alternative to the Harris RTXDS. The documentation and the board's design alone will pay for the system in the time they save the engineering staff. The software development tools aren't too shabby either.

We're Backing You Up With Products, Support, And Solutions!

Signal Processing
- Linear
- Custom Linear
- Data Conversion
- Interface
- Analog Switches
- Multiplexers
- Filters
- DSP
- Telecom

Digital
- CMOS Microprocessors and Peripherals
- CMOS Microcontrollers
- CMOS Logic
- CMOS Memories

ASICs
- Full-Custom
- Semicustom
  - Gate Arrays
  - Standard Cell
  - Cell Based
  - Core Processors
- ASIC Design Software

Power Products
- Power MOSFETs
- IGBTs
- Bipolar Discretes
- Transient Voltage Suppressors
- Opto Devices
- Power Rectifiers

Intelligent Power
- Power ICs
- Power ASICs
- Hybrid Programmable Switches
- Full-Custom High Voltage ICs

Microwave
- Gallium Arsenide FETs
- Standard MMICs
- Custom MMICs

Military/Aerospace Products
- Microprocessors and Peripherals
- Memories
- Analog ICs
- Digital ICs
- Discrete Power
  - Bipolar
  - MOSFET
  - Rad-Hard ICs

Military/Aerospace Programs
- COMSEC Programs
- Strategic and Space Programs
- Military ASIC Programs

SALES OFFICE HEADQUARTERS

United States
Harris Semiconductor
1301 Woody Burke Road
Melbourne, Florida 32902
TEL: (407) 724-3739

European
Harris Semiconductor
Mercure Centre
Rue de la Fusse 100
Brussels, Belgium 1130
TEL: (32) 246-2201

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

South Asia
Harris Semiconductor H.K. Ltd.
13/F Fourseas Building
208-212 Nathan Road
Tsimshatsui, Kowloon
Hong Kong
TEL: (852) 3-723-6339

HARRIS SEMICONDUCTOR
HARRIS · RCA · GE · INTERSIL

Reorder Number: AR-RTX-002

HARRIS SEMICONDUCTOR

Reprint Series

RTX Family Benefits for Real-Time Control Applications

TABLE OF CONTENTS

HARRIS

Stack-Based Processor Speeds Target Recognition

Chip executes all instructions in a single machine cycle

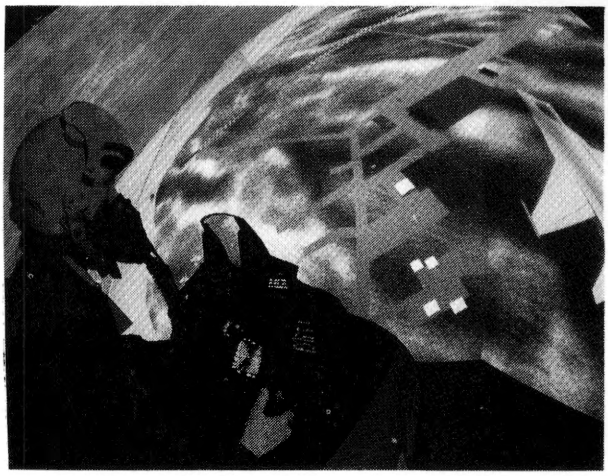
David J. Bak, East Coast Editor

Melbourne, FL—You're flying an F-16. Infrared sensors suddenly lock onto an object approaching at 1000 mph. Friend or foe? You've got 10, maybe 20 msecs to extract an edge, identify, and respond. The almost-instantaneous code execution needed to perform such a task demands a processor that addresses events with a predetermined, repeatable, sequence of operations.

Most processors sold today boost speed at the expense of predictability. Based on RISC (Reduced Instruction Set Computer) architectures, they assign data and instructions to separate on-chip register banks, i.e., sets of flip-flops clocked by the processor and used for storage. These register banks, in turn, permit the simultaneous execution of independent operations using simple fetch-and-store commands. Good for straight-line program code, the architecture presents problems when used in real-time control applications.

Their ability to rapidly respond to internal and external interrupts, for example, depends on the state of the individual registers when the interrupt is acknowledged. Before servicing the subroutine, the processor must "clean itself out," transferring everything in the registers to the main memory. This process, called context saving, slows execution time. Worse yet, because the processor is never in the same state when the interrupt occurs, execution times are unpredictable—a condition that's often unacceptable (as seen in the F-16 scenario).

A new processor, based on four parallel internal buses, reflects the most desirable features of an ideal RISC processor. Its internal data paths are directly routable to the ALU (arithmetic logic unit) in a single machine cycle, with no sacrifice in execution predictability. And, with the exception of memory reference instructions that consume two machine cycles each, the processor executes all of its instructions in a single clock cycle. Like RISC, instructions are decoded in logic, rather than translated into a sequence of microcoded primitives. Unlike RISC, context saving is not necessary. The processor does not discriminate between direct or branched flow, treating interrupts as simple subroutine calls.



Crucial to avionics target-tracking, high-speed embedded real-time systems can perform single-target extraction operations in as little as 10 msecs—an amount of time in which most general purpose processors can only execute a maximum of around 100 instructions. (Photo: ©Fred Ward, Black Star.)

Built by Harris Corp., the RTX-2000 includes a main memory bus, a high-speed, bidirectional Application Specific Integrated Circuit (ASIC) bus, and two on-chip stack-memory buses. The main memory bus carries the sequence of instructions needed to execute a given task. Additionally, main memory holds all data not needed for on-chip storage. The ASIC bus allows easy interface with application specific ICs to further enhance system processing throughput.

Two parallel on-chip LIFO memories—a parameter stack and a return stack—take care of internal storage. The parameter stack provides storage for all operands. Addresses of all subroutine returns are stored in the return stack. Use of the two-stack system minimizes overhead associated with procedural calls, branches, and jumps. In addition, this approach reduces programming complexity.

Due to its high level of parallelism, the RTX-2000 can execute up to five FORTH primitives concurrently within a single, 16-bit machine instruction cycle. Consider the sequence of instructions involved in the last cycle of an image-processing task, such as edge extraction. The processor can:

- Read a 16-bit data value from the ASIC bus.
- Perform an ALU operation.
- Store the result on the parameter stack.

- Perform a subroutine return.
- Fetch the first instruction of a subroutine returned to by the processor.

Comparatively, such concurrent execution would equate to three or four RISC-equivalent instructions. On most RISC processors, executing that many instructions would require 6 to 8 bytes of program code and as many as 16 clock cycles.

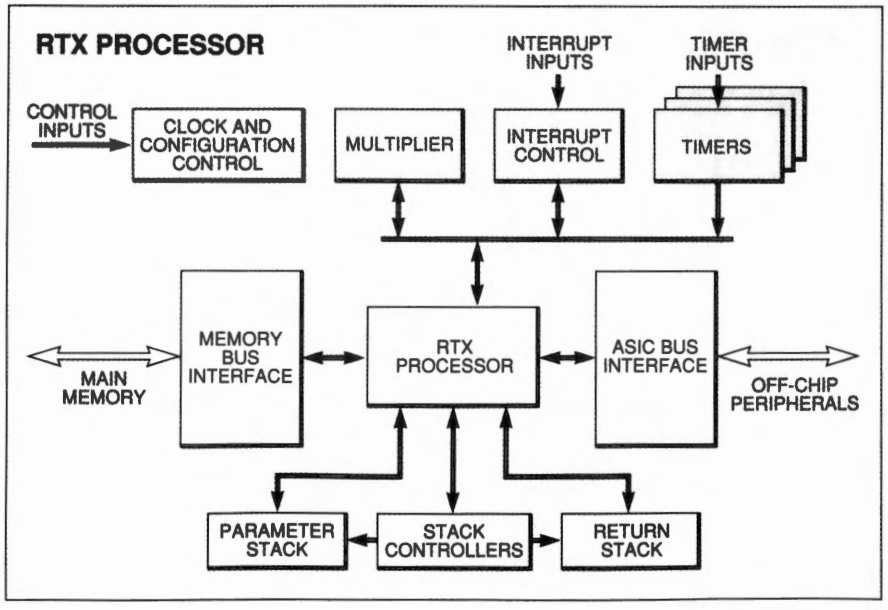
Stack controllers, timers, a multiplier, and an interrupt controller complete the on-chip package. Stack controllers internally maintain the absolute address of the stack location's current 16-bit "top" value, onto which the next datum will be pushed. This value, stored in the controller's stack-pointer register (SPR), can be read by the processor at any time, such as during the process of switching a task context. The SPR can also be written into in the process of restoring the context of a new task.

Implementation of the interrupt controller allows 14 prioritized interrupts. Of these, nine are used by the processor's internal peripherals. The remaining five are available as inputs to the processor. The RTX-2000's on-chip 16- × 16-bit parallel multiplier functions as the processor's major source of arithmetic power, while three on-chip, 16-bit down-counters perform all timing and event-counting tasks.

As an integral part of the Ball Aerospace Daylight Star Tracker, the RTX 2000 processes 900 CCD pixels in 9.1 msecs. The Tracker, described on page 132 of the May 22, 1989 issue of *Design News* measures the bearing and azimuth of a star to determine the point on the earth's surface occupied by the observer. Other applications, besides image processing, avionics and space guidance systems, include robotic motion/effector control, speech processing, smart munitions, and all areas of graphics processing. Embedded expert systems is another area of potential use.

Additional details . . . Contact David Williams, Harris Corp., Semiconductor Products Div., Box 883, Melbourne, FL 32902, 407-729-4629. □

Internal, highly parallel busing gives the RTX-2000 extremely high throughput capability. The only class of two-cycle intructions needed are memory-reference instructions, i.e., one cycle to fetch an instruction, and one to fetch/store the data.



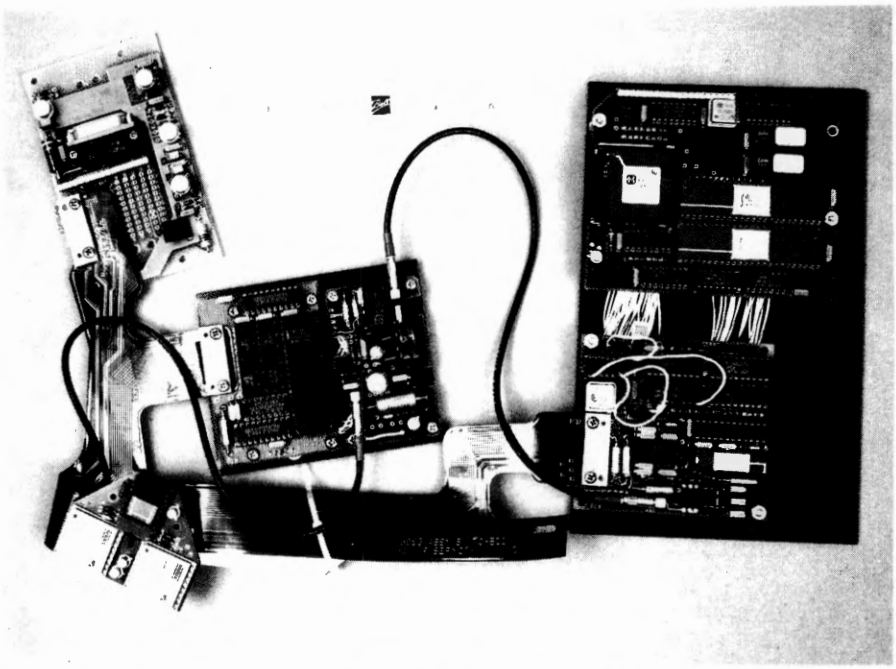
High-Speed Microprocessor Makes Star Tracker Functional

Unit processes 900 pixels in 10 msecs and consumes just 3.1W

Lyle H. McCarty, Western Editor

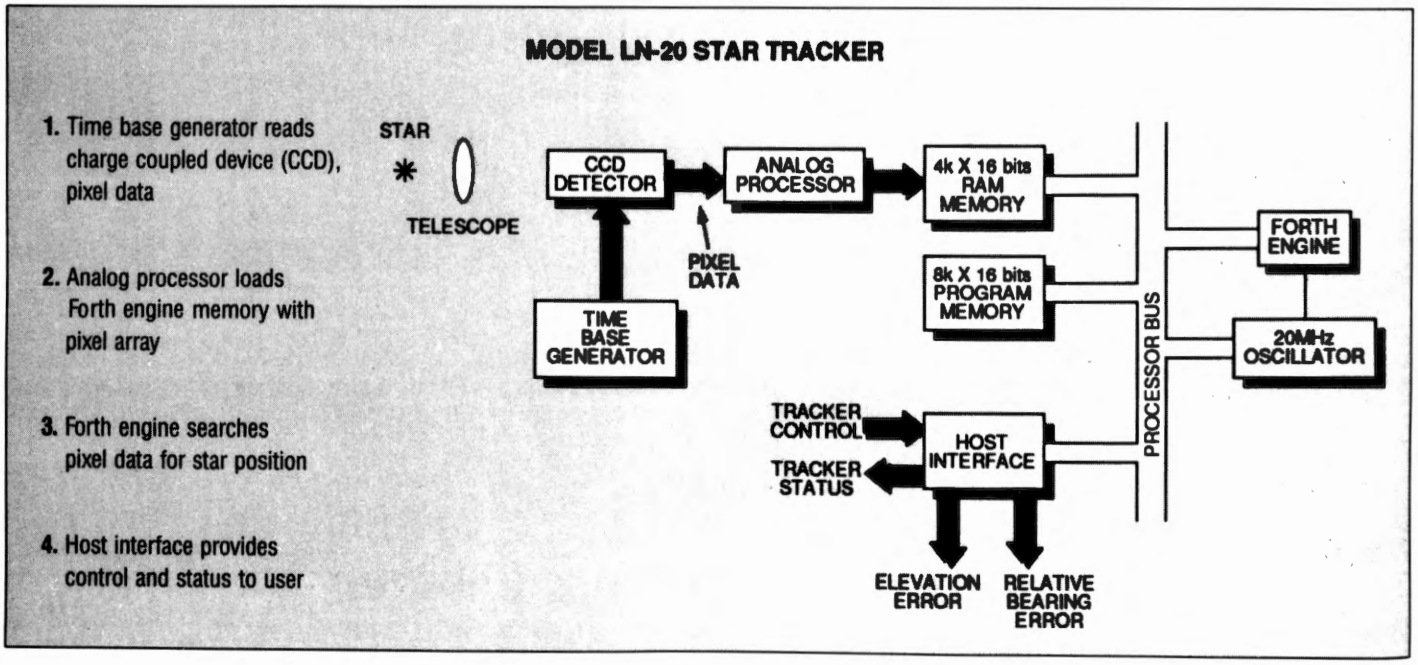
Boulder, CO—Star trackers do what navigators did in days of yore—measure the bearing and azimuth of a star to determine the point on the earth's surface occupied by the observer. That information is then put to work—correcting the navigator's dead-reckoning calculations in the old days, providing updated zero references for the inertial navigation system today.

Of course, star trackers work much more quickly than the busiest of navigators, so fast that the equipment shown here was required to process signals from 900 Charge Coupled Device (CCD) pixels in 10 msecs. Conventional multiclock cycle processors can execute only five to ten instructions in this time interval, not nearly enough computing power for the job.



Star tracker electronics assembly occupies little space, weighs just 14.2 oz, and operates on 3.1W. Its Harris microprocessor processes 900 CCD pixels in 9.1 msecs.

To solve this problem, Ball Aerospace engineer Mike Hubbard turned to a Harris Corporation microprocessor known as a Forth en-



gine. This device runs at a clock speed of 10 MHz and can execute more than 100 instructions/msec on each pixel. It is also small—one-inch square by ¼-inch high—and uses just 25 mW/MHz. In addition, the chip requires only a small amount of support circuitry to form a complete computer system.

Tracker electronics are functionally partitioned into camera and processing areas. The camera is subdivided into a CCD focal plane, a correlated double sampler analog processor, and a Time Base Generator (TBG). A state machine sequencer running at 12 MHz, the TBG provides CCD control signals to read out the 900 pixels in 10 msecs. This clever circuitry includes three programmable array logic devices and a 32k × 16 bit EPROM. The waveform required to read out the CCD is determined by the bit patterns that are burned into the EPROM.

A camera interface, dual D/A converters for X and Y analog output error signals, and digital input and output ports comprise the analog processor. The unique camera interface loads pixel data into RAM with no processor overhead. Major components of the interface are a 12-bit A/D converter, dual-port 4k × 16 bit RAM, and counter. While the A/D converter converts data and writes it into RAM, the counter provides RAM addresses.

The Forth Engine accesses pixel data on the other RAM port to determine star position. The RAM also provides variable and array data storage. Constants and the star tracker program are stored in an 8k × 16 bit EPROM. A host interface enables a user to determine system status and control the star tracker.

Additional details . . . Contact Robert Baker, Ball Aerospace Systems Division, Box 1062, Boulder, CO 80306-1062, 303-939-4917. □

EMBEDDED CONTROLS IMPROVE EXERCISE FOR HANDICAPPED

Advanced microprocessor triggers electrical stimulation of paralyzed muscles in real time

Gail M. Robinson, Associate Editor

Like many of us who are listening to the advice of our doctors, Arthur Schlesinger works out three times a week. Taking about 30 minutes, his workout involves a complete cardiovascular program.

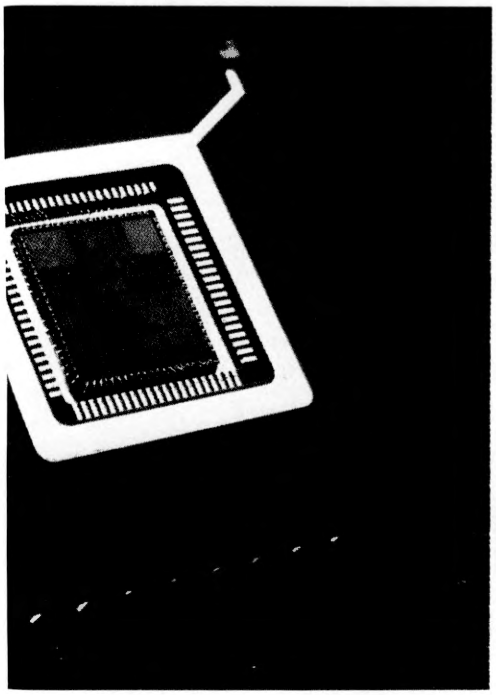
But there is one major difference. Schlesinger is a paraplegic. He is a volunteer in a program at Wright State University Medical School, Dayton, OH, geared to developing complete exercise programs for handicapped people. Though he has no use of his legs, Schlesinger gets an aerobic workout by riding a bicycle while turning a gear crank.

The key to the workout is a system that controls electrical stimulations which, when applied to Schlesinger's legs, produce functional muscle contractions. The controller is a new type of microprocessor that takes on the normal stimulus and response functions of the nervous system. Its job is to synchronize the lower and upper extremities so the individual gets all the benefits of a cardiovascular workout.

Called the RTX 2000, this high-speed 16-bit programmable microcontroller may prove to be a major contender in the microprocessor market for real-time applications. Designed by Harris Semiconductor, Melbourne, FL, the system offers interrupt response times of 400 nsecs, context switch times of 2 μsecs, and an instruction execution

ERGYS I bicycle aids indivduals who do not have control over the use of their limbs. The system is used with a computer to control electrical stimulation that produces muscle contraction, enabling the individual to pedal.

rate of over 10 MIPS. And, it features a new way of dealing with real-time events. Instead of a simple interrupt, it can merge responses from different channels to make up a unique set of instructions that handle more than one instruction at a time.



The RTX 2000 is a sophisticated microcontroller optimized for real-time applications.

Fitting a need

"Everyone knows that without activity the muscles go through atrophy," says Bertram Ezenwa, director of technical development in the Div. of Research and Development at Wright State. "Unfortunately, people with paralyzed limbs can't do anything about it. Their muscles go through complete deterioration. Not only are the physical problems difficult to deal with, but there are psychological problems as well."

Ezenwa's goal is to design a system that gives these people the opportunity to be able to do a complete workout, rather than exercise only certain parts of the body. "The control system plays a major role in this kind of exercise program," says Ezenwa. "It must determine the condition of the lower extremity musculature, for example, the fatigue condition, and the phase and rate of the arm cranking. It then applies the necessary current in real time to specific lower extremity muscles so they are synchronized with the arm cranking."

Most conventional microprocessors are optimized for office computers or computer-aided workstation environments. In these cases a "real-time" response may take a few seconds, probably making little difference to the user.

However, these processors are not designed for applications where even a ½-sec response time to an external event is too slow. When dealing with more than one task, a conventional microprocessor must jump back and forth between the different instruction groups.

For example, in this case one set of instructions is set for pedaling the bicycle, while a completely separate set of instructions deals with the turning of the crankshaft. Though both of these tasks must occur simultaneously, a conventional microprocessor can only process one set of instructions at a time in the sequence. It must jump from one task to another, interrupting the other task in the process. If the controller is well-matched to the job, it can jump back and forth fast enough to look as though it were doing both tasks simultaneously. But the chance of an overload is very high.

With the RTX 2000, two different sets of instructions for each task are not required. Instead, the microcontroller makes one set of instructions that deals with the two different events. Therefore, it deals with a greater variety of tasks using the same level of processing power.

Taking a closer look

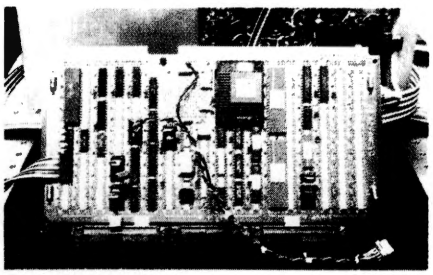
"There is a growing need for processsors with higher performance standards than existing microcontrollers," says Dave Williams, manager of the RTX product marketing at Harris. "Even RISC processor technology with its promise of adding even more sophisticated products to the marketplace may not be suitable for many embedded control applications."

Though RISC processors are making strides in real-time processing, especially in respect to speed and performance, Williams notes that most real-time system environments are driven by asynchronous external events, which require fast interrupt response and predictable system level timing.

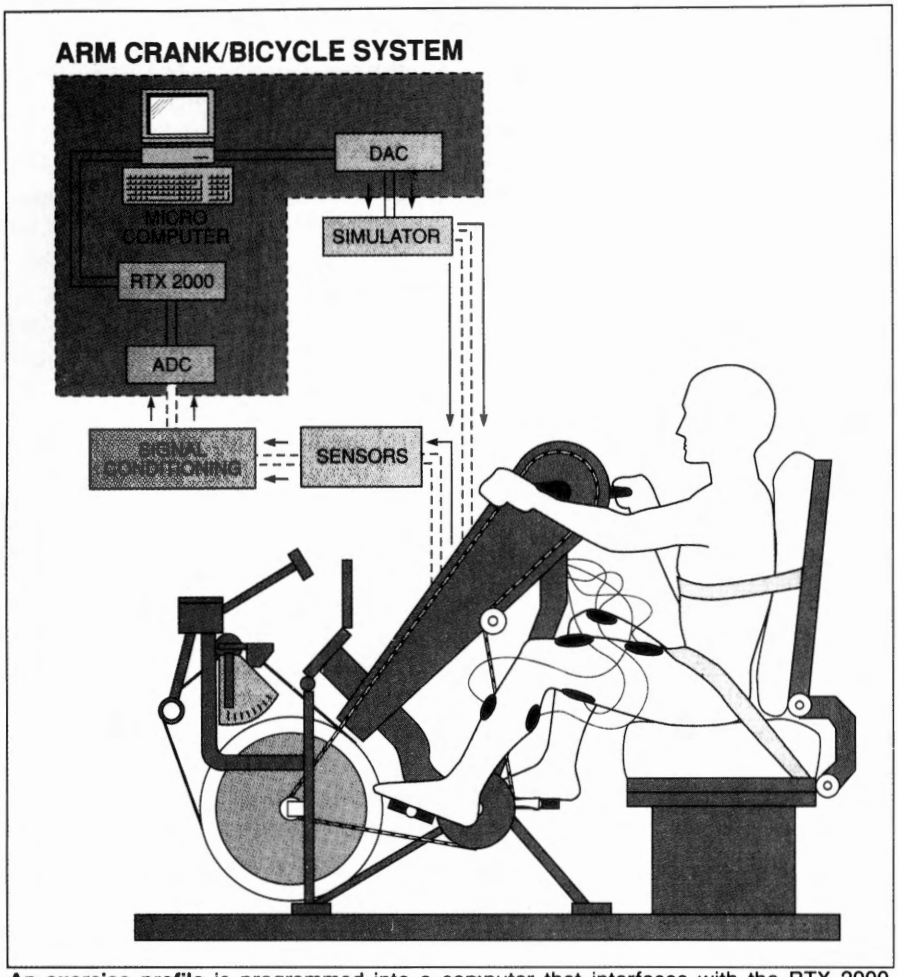
Williams adds that RISC processors rely on pipelining and cache memory schemes to provide improvements in the statistical speed of the processor, so response times are often unpredictable.

Though the RTX does exhibit RISC-like behavior in its performance of one instruction per cycle, there is an important difference in how the data and control paths are configured. The microcontroller does not use pipelines to achieve its speed, but instead it uses parallel techniques in the form of a RTX Quad Bus. This enables several different kinds of information transfer to occur at the same time as part of one instruction.

The chip also has no microcode



Space Shuttle uses RTX 2000 in solid-state Star Tracker device.



An exercise profile is programmed into a computer that interfaces with the RTX 2000. The controller accesses the required muscle contractions and synchronizes the lower and upper extremities of the body.

sequencer or microcode. All instructions except memory access instructions execute in one cycle. The system minimizes address calculation delays by incorporating a simplified memory paging mechanism, and eliminates the complexity of multiple addressing modes and memory arrangement.

The RTX is also a stack machine. Stacks facilitate the evaluation of expressions and minimize the control overhead needed to organize data. A stack machine not only uses a stack for temporary data storage, but executes all operations on data from the stack. Thus, the ALU finds all of its data in a pre-defined location, and can get that data without an address specification.

Besides two 256-word stacks, the chip has several other features that may keep it ahead of the pack, including the interrupt controller, three on-chip 16 bit timer/counters, an ASIC bus for off-chip extension of the architecture, and 1M byte of memory address space.

Putting it to work

So far, parts of the real-time data acquisition and display for the system have been tested on an Intel 386-based microcomputer. "After we applied the RTX 2000 real-time operating system, the data acquisition and display were much faster," says Ezenwa.

To initiate the program, an exercise model is programmed into the microcomputer. The controller then accesses the required muscle for contraction through surface electrodes attached to the body. The model or alogorithm is downloaded in FORTH from the microcomputer to the RTX chip.

An integral part of the equipment is a specialized stationary bicycle designed by Therapeutic Technologies Inc., Dayton, OH. The ERGYS I is designed to combine functional electrical stimulation with such advances in microprocessor development to control the muscle contractions. The system on the market now is directed towards home rehabilitation. It features a programmable electronic cartridge for individualized treatment.

For Schlesinger's particular workout, the system stimulates three different muscle groups: the quadriceps, the hamstrings and the gluteus maximus. This enables him to ride the stationary bicycle as he turns the arm crank. After 36 sessions, he undergoes strength, muscle, and cardiovascular testing.

"The physical results have been great," says Schlesinger. "My muscle mass and strength have increased, I have more circulation, and noticeably less fat. But the psychological effects are just as important. My muscles have tone and look normal. I can't tell you how good that makes me feel. I look better and I feel better about myself."

Researchers are currently developing simulators for selective stimulation of deep muscles from surface electrodes. And, Dr. Ezenwa is also designing a system for weight lifting. "By applying the proper amount of electrical current to the quadricep muscles, the leg can lift weight against gravity," says Ezenwa. "But several conditions must be factored into the control design, such as applying only the necessary current intensity for each phase of contraction, determining muscle fatigue condition, and keeping the weight lifting to predetermined safe limits."

To follow these parameters, Dr. Ezenwa is developing an automated stimulator for the adaptive control of knee extension exercise for spinal cord individuals. The RTX 2000 development system will carry out the real-time control algorithms.

Beyond the health field

There are other research applications in need of such systems as well, including many outside the medical field. Fred Sias, professor in the electrical engineering dept. at Clemson University, Clemson, SC, is designing wheeled mobile robots for surveillance and radiation environments.

He expects to apply the RTX 2000 "because it offers optimized control for multitasking and interrupts, and it is easy to interface with the real world."

Ball Aerospace, Boulder, CO, is using the RTX 2000 in its newest solid-state Daylight Space Tracker. The Tracker is an electroptical device used in aircraft and the space shuttle that analyzes stars through a scope. The signals it generates control the body surrounding the spinning wheels of the gyroscope. A computer responds to the Tracker's signals and moves a platform to keep the Tracker aligned.

The system mounts on a guidance gimbal and uses a CCD sensor and the RTX 2000 to process a 900 pixel frame in 10 msec. The RTX scans the field and locates the stars, and then sends the tracking information to the computer.

Exceptional instruction speed was the main reason the company is using the chip, notes design engineer Mike Hubbard. "I doubt if the job would have been done without the RTX," he explains. "If so, it would have had to be custom-made, which would have been more expensive and taken more time." □

We're Backing You Up With Products, Support, And Solutions!

Signal Processing
- Linear
- Custom Linear
- Data Conversion
- Interface
- Analog Switches
- Multiplexers
- Filters
- DSP
- Telecom

Digital
- CMOS Microprocessors and Peripherals
- CMOS Microcontrollers
- CMOS Logic
- CMOS Memories

ASICs
- Full-Custom
- Semicustom
  - Gate Arrays
  - Standard Cell
  - Cell Based
  - Core Processors
- ASIC Design Software

Power Products
- Power MOSFETs
- IGBTs
- Bipolar Discretes
- Transient Voltage Suppressors
- Opto Devices
- Power Rectifiers

Intelligent Power
- Power ICs
- Power ASICs
- Hybrid Programmable Switches
- Full-Custom High Voltage ICs

Microwave
- Gallium Arsenide FETs
- Standard MMICs
- Custom MMICs

Military/Aerospace Products
- Microprocessors and Peripherals
- Memories
- Analog ICs
- Digital ICs
- Discrete Power
  - Bipolar
  - MOSFET
  - Rad-Hard ICs

Military/Aerospace Programs
- COMSEC Programs
- Strategic and Space Programs
- Military ASIC Programs

SALES OFFICE HEADQUARTERS

United States
Harris Semiconductor
1301 Woody Burke Road
Melbourne, Florida 32902
TEL: (407) 724-3739

European
Harris Semiconductor
Mercure Centre
Rue de la Fusse 100
Brussels, Belgium 1130
TEL: (32) 246-2201

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

South Asia
Harris Semiconductor H.K. Ltd.
13/F Fourseas Building
208-212 Nathan Road
Tsimshatsui, Kowloon
Hong Kong
TEL: (852) 3-723-6339

HARRIS SEMICONDUCTOR
HARRIS · RCA · GE · INTERSIL

Reorder Number: AR-RTX-001

HARRIS SEMICONDUCTOR REPRINT SERIES

Editorial Coverage by Leading Industry Publications

FORTH Processor Core for Integrated 16-Bit Systems

Peter S. Danile and Christopher W. Malinowski, Harris Corp. Semiconductor Division, Melbourne, FL

The development of a high-performance dedicated 16-bit processor using an industry-standard microcontroller or bit-slice processor calls not only for an extensive board-level design effort, but also for a long-term development program for software and firmware. It has been difficult to use semicustom techniques for such development because core processors have been scarce and microcode development for custom ICs is very difficult.

However, in a growing number of high-performance systems for digital signal processing, control, and arithmetic, application-specific processors are bringing forth the advantages of integration—including high throughput, low power dissipation, and much higher density than board-level implementations.

Harris Semiconductor now has a semicustom technology, called the Processor Toolbox, that eases the development of high-performance 16-bit integrated processors. Toolbox features include a very small core-processor cell, a highly parallel architecture for maximum throughput, easy programming and code development, code portability, a full set of core-compatible peripheral cells that can support processor clock frequencies as high as 15 MHz, and a set of high-speed arithmetic and logic cells, such as a 16-bit multiplier, for further customization.

The processor architecture derives from one conceived by Charles Moore, inventor of the FORTH language. This RISC-like, highly parallel architecture meets the size and throughput requirements. The combination of the processor's instruction set—a directly executable set of FORTH high-level primitives—and its reliance on two stacks that reflect a FORTH virtual machine results in a compact core processor with less than 2500 gates. This core is called the FORTH Optimized RISC Computing Engine, or Force.

Because each instruction comprises more than one FORTH primitive (opcode), data-manipulation throughput can exceed the processor's clock frequency, often by a factor of three. Consequently, for instruction sets rich in multiple-opcode instructions, peak processing throughput can exceed 30 MIPS, with a steady throughput of 10 to 20 MIPS.

Users of the Toolbox can develop application code in a high-level FORTH language working with an interactive and interpretive environment. FORTH is not only portable but also offers expeditious debugging tools and easy target-compilation from one environment to another. Therefore, a wealth of code written for DSP, artificial intelligence, control, number-crunching, and real-time data-processing applications can be ported to the Harris engine.

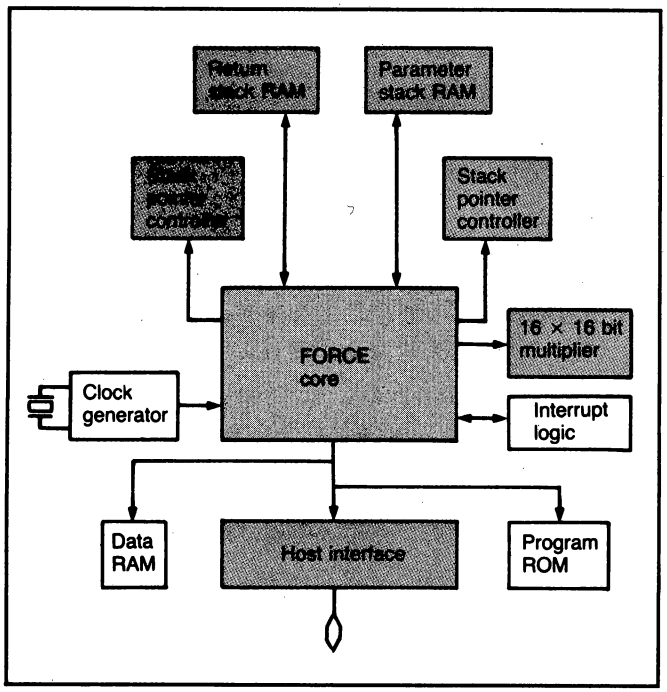


FIGURE 1. The Force Toolbox contains a FORTH processor and support peripherals, all available as cells and packaged parts.

The principal advantage to designers of dedicated processors stems from the host of proprietary LSI and VLSI cells being developed to support the FORTH core processor. The Toolbox provides designers of Force-based products with a set of packaged Force circuits identical to the cells available in Harris' standard-cell library (Figure 1). The designer can use these parts to create a breadboard and experiment with different configurations of the Force processor before moving to an ASIC. He gains greater confidence in the design's functionality, because it can be exercised in a real environment in addition to a CAE simulation environment. Moreover, the designer can use the prototype breadboard to generate a reliable and accurate set of functional test vectors, a task both formidable and error-prone otherwise.

Another advantage of creating a breadboard is that the designer can compile the applications code and run it before committing it to a ROM pattern. Running the application code lets the designer make trade-offs between implementing func-

Reprinted from VLSI Systems Design June 1987, Copyright 1987 C.M.P. Publications Inc.

tions in hardware and coding them in software. The core's execution speed allows many functions—such as memory swaps, arithmetic and logic functions, shifts, and masking—to be implemented in firmware without sacrificing performance, shrinking die size considerably. Such trade-offs can be investigated only if the designer has access to the processor's functional blocks and can exercise alternatives in real time—that is, on a breadboard.

Finally, the Toolbox approach makes it easier to design a testable circuit. During breadboarding, the designer can observe the timing of signal paths, such as the processor's data paths, which may not be directly accessible in an integrated system. Identifying buried trouble spots increases the understanding of the circuit's testability requirements and makes it easier to implement such testability features as scan paths and memory-check routines.

The Toolbox version of the core processor (Figure 2) comes in a 144-pin pin-grid array with all the I/O signals bound to pins. Besides the core, the Force Toolbox also contains packaged versions of a stack controller with an on-chip 64×16-bit stack RAM, an interrupt controller, and a 16×16-bit multiplier. By early 1988, Harris also will offer a hardware-development system and a Force target compiler that, in combination with a breadboard system, will support firmware-code development.

Once hardware and firmware are defined, the design is implemented in a semicustom IC, for which the designer customizes the program, data, and stack memories by using RAM and ROM module compilers. As much as 64K of on-chip firmware ROM can be integrated in addition to 16K of data RAM. Because FORTH code is so compact, very extensive application-specific code can be implemented in firmware along with the kernel code. To replace the discrete logic on the breadboard, the designer uses 7400-type SSI and MSI cells from the Harris standard-cell library.

The Force Core

The Toolbox's Force core processor is a bare control engine with 123 I/O lines. These signals include three parallel 16-bit data buses (two for stack memories and one for main memory), a 16-bit main-memory address bus, and a dual-purpose 5-bit address-extension bus. In addition, a general-purpose 16-bit bus (G-bus) acts as the processor's primary I/O signal path.

A principle of RISC philosophy is to bring execution speed as close as possible to the maximum memory-access speed. As a corollary, the number of multicycle instructions in the processor's instruction set is reduced to maximize the processor's data-bus throughput and bandwidth.

The processor's independent buses account for the Force core's high throughput. Because each instruction executes in no more than two clock cycles, at least three of the five buses are active during any clock cycle. The G-bus transfers data at up to 30 MB/s when the processor operates at 15 MHz; the main-memory bus transfers data at up to 30 MB/s when in a streamed-move mode.

The Force core processor's highly parallel architecture (Figure 3) reflects the structure of its horizontal instructions. Its eight main registers provide parallel storage and access to the parameter stack's top two locations (TOP and NEXT), the top location of the return stack (I), the instruction register

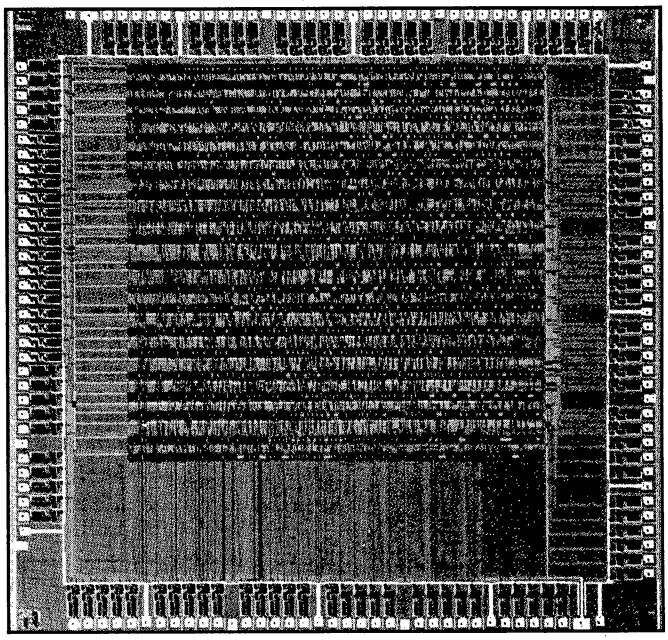


FIGURE 2. The packaged version of the Force core processor.

(IR), the program counter (PC), and two arithmetic-instruction registers (MD and SR). The MD register stores the partial results of step-multiply and step-divide instructions; the SR register stores the partial results of hardware-assisted square-root operations.

The minimal overhead to support subroutines also reflects the nature of the FORTH language, which is heavily oriented toward the use of subroutines. The core processor is optimized for the minimum number of cycles necessary to execute a subroutine call and return. Through instruction partitioning and architectural refinement, the execution of a subroutine call requires only a single clock cycle; the return requires no added clock cycles. All interrupts that are interpreted as subroutine calls therefore require only one clock cycle of overhead.

Stack Controller Cell

Because the stack-oriented FORTH instructions employ stacks in every command, the most critical peripheral used with the Force core processor is the stack controller. Controlling the stacks with software routines would degrade the system's performance. The core directs the stack controller through the RW and SA signals. The stack controller responds to the SA signal with either a push (data write) or pop (data read), according to the status of the RW signal.

In the packaged version of the stack controller are 64 words of memory with an access time of approximately 30 ns, so the designer can operate the core at 15 MHz without worrying about the access time of data in the stack. In an ASIC implementation, the stack's memory size can be altered and the access time improves to about 20 ns.

The stack controller operates with the core processor's parameter stack (through SAS and RWS signals) and the return stack (SAR and RWR), giving a typical system two stack controllers (Figure 4). To enhance system flexibility, the signals OVER and UNDER generate interrupts when the stacks are ready to overflow or underflow. UNDER occurs when the

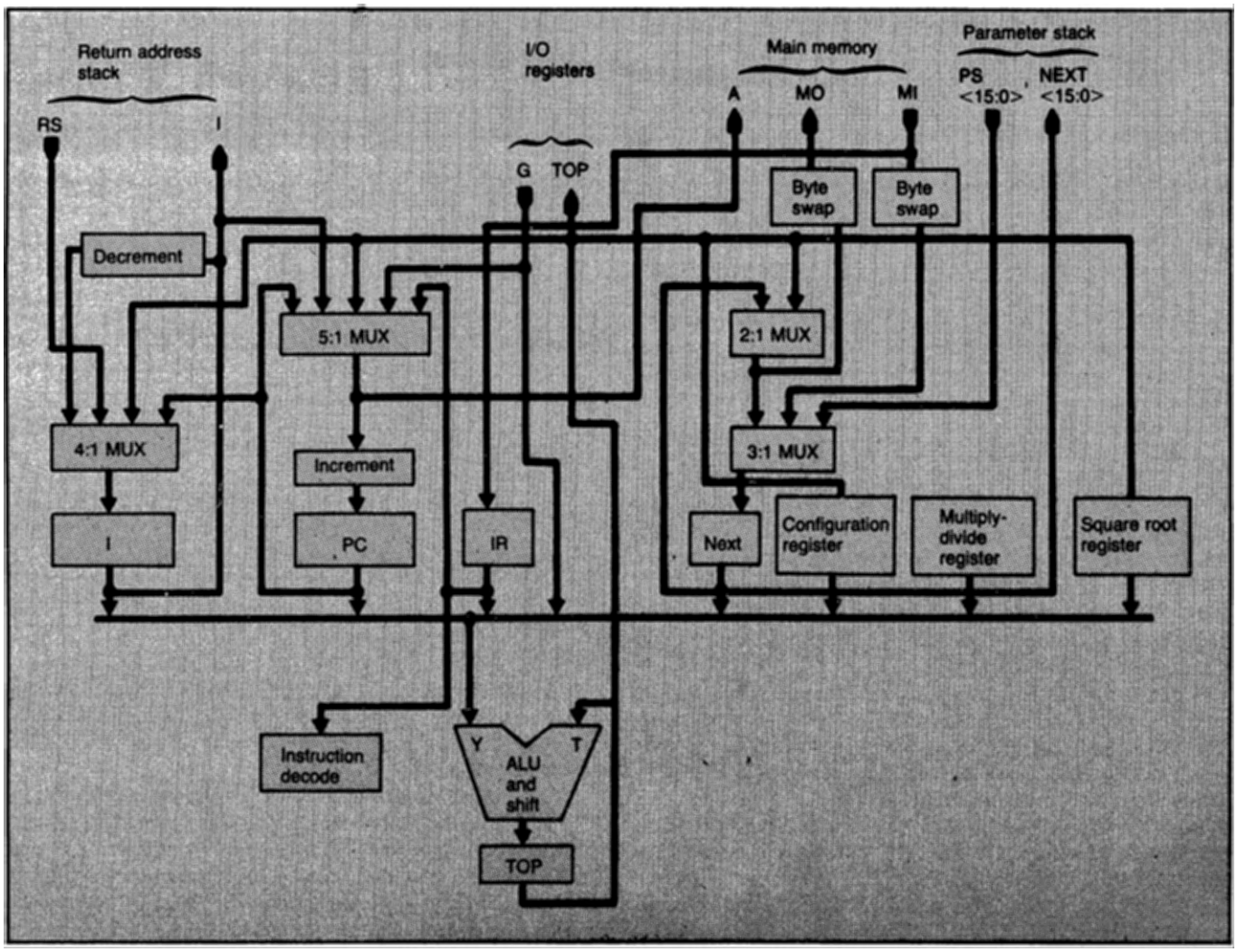


FIGURE 3. Parallel architecture lets three of five processor buses be active for all instructions.

stack is pushed more than popped and the stack address lines reach 00. The assertion should initiate a routine that either resets the controller (if more returns than routines were called) or tries to recover from other sources of underflow. The OVER signal occurs when the number of words pushed onto the stack exceeds a user-defined maximum. This maximum can be programmed into the ASIC implementation by writing into an offset register through the G-bus.

To implement multitasking versions of the Force processor, Harris is developing a multitasking stack controller (MSC). The MSC enables the user to partition the 256-word stack into eight separate stacks via a 3-bit address size register. For example, if the system must run two concurrent jobs, the size register is programmed to divide the stack RAM into two 128-word stacks. To switch between tasks, the processor enables a task-select register through the G-bus. When this register is written to, the stack pointer of the current task is saved and the stack pointer for the new task is restored at the new task's current address.

When the stack controllers and core processor are integrated on a single chip, options for increasing performance are available. Not only can the access times of the controller and the data RAMs be reduced merely by integrating, but access time can also be reduced further by separating the bidirectional data buses into read (POP) and write (PUSH) buses (Figure 5). In the discrete versions, the data buses are bidirectional to allow them to connect directly to standard RAMs with bidirectional data buses. Separating the buses adds some additional routing area; on the other hand, the time required to set up the buses for either a read or a write is eliminated, improving system response time substantially.

Interrupt Controller and Host Interface

The interrupt controller and the host interface help the core processor interact efficiently with the surrounding system. First, the interrupt controller contains 15 prioritized interrupt-request inputs and a separate input for nonmaskable interrupts (NMI). The interrupt priorities are fixed (to decrease response time) but can be defeated by writing in the interrupt controller's mask register (which has a discrete address on the G-bus). The interrupt controller samples the request inputs on opposite edges of the system clock. When two consecutive samples confirm that an interrupt is present, the INT line is asserted. The core processor responds with the INTA signal, which directs the interrupt controller to generate the appropriate vector for the interrupt. This vector comprises a 7-bit user-defined field for the location of the interrupt-vector table and a 5-bit field that designates the appropriate interrupt.

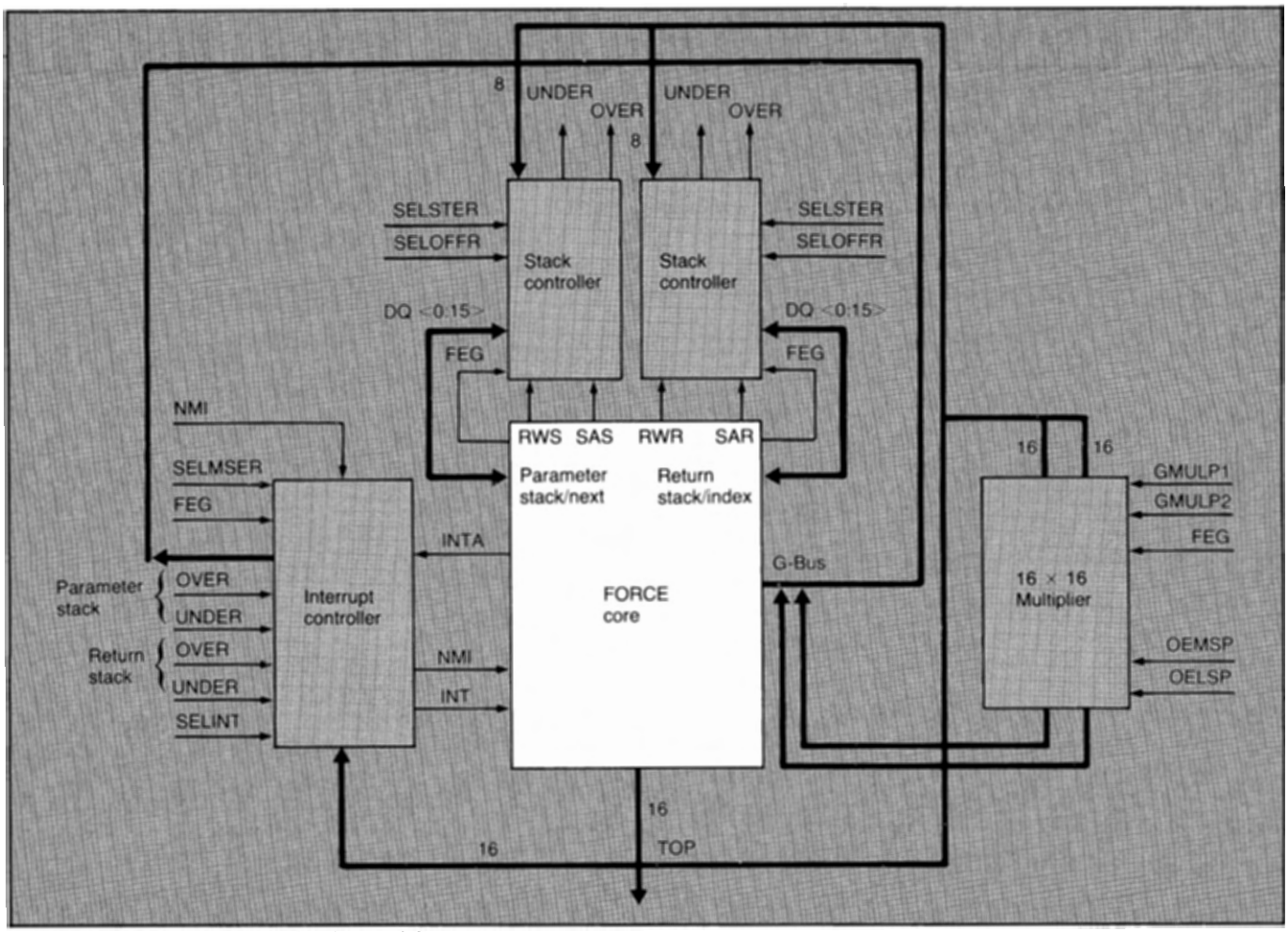


FIGURE 4. Breadboard design of Force system.

In the packaged version of the interrupt controller, the interrupt vector for the valid highest-priority interrupt is presented to the core processor within 40 ns of the INTA pulse. Thus the controller can operate with the core processor at system frequencies greater than 15 MHz. Once the system has entered an interrupt routine, the core's INTE signal inhibits any new interrupt from generating a new INTA. During INTE assertion, the system can clear the interrupt source and rewrite the Force configuration register to re-enable the interrupts. The interrupt controller does not automatically nest interrupts, so the system does not need to modify the interrupt controller until it must mask or unmask any interrupt line.

To create an interface between the Force core and a host controller that does not degrade the core's performance, the Toolbox includes a host interface cell. The interface allows another processor to read and write data in the Force core's memory-address space; it receives as inputs the address and data lines of the shared memory, the command lines from the processors, the core's data signals, and a MEMORY_READY signal that indicates when the data in the shared memory is valid. It provides a READY signal to the host, to indicate when it can read or write data, and a clock signal to the core. Figure 6 shows a typical configuration of the interface, the core processor, and the shared memory.

The host interface suspends the core processor when it attempts to read invalid data. When the data is not in the shared memory, the MEMORY_READY input to the host interface is de-asserted until the data in the memory becomes valid. While the signal is low, the interface suspends execution by the core processor; the processor continues when MEMORY_READY is re-asserted.

When the host processor wants to write to the shared memory, the host interface checks the FORCE_LOCK signal to determine if the Force processor has priority on the memory bus. If not, the interface suspends the core processor and hands control over to the host. If wait states are necessary, MEMORY_READY becomes low and the host interface stalls the host with the HOST_READY signal. When HOST_READY is asserted, the host can relinquish priority on the bus.

Writing host-processor data into the shared memory is simpler than reading from it. The host writes directly to the host interface, which holds the data in a buffer. When the memory bus becomes free, the interface writes the data into the memory. A HOST_READY signal is set when the buffer is full to prevent the host from writing over new data.

If the host processor wants to do some house-cleaning in the shared memory, it requests priority on the memory bus by asserting the HOST_LOCK pin. This signal suspends the core processor so the host can have exclusive access to the memory. In normal operation, the use of LOCK signals should be minimized so performance is not degraded.

The host interface is designed to work with an asynchro-

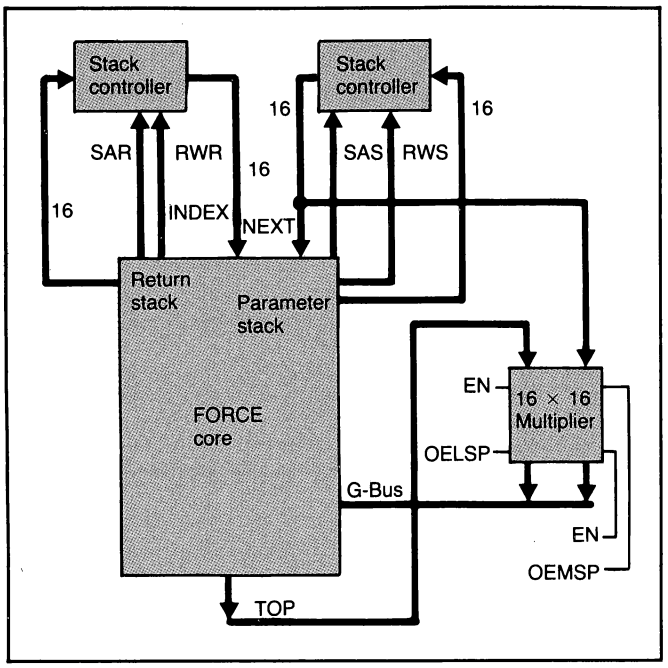


FIGURE 5. Integrated configuration of core, stack controller, and multiplier.

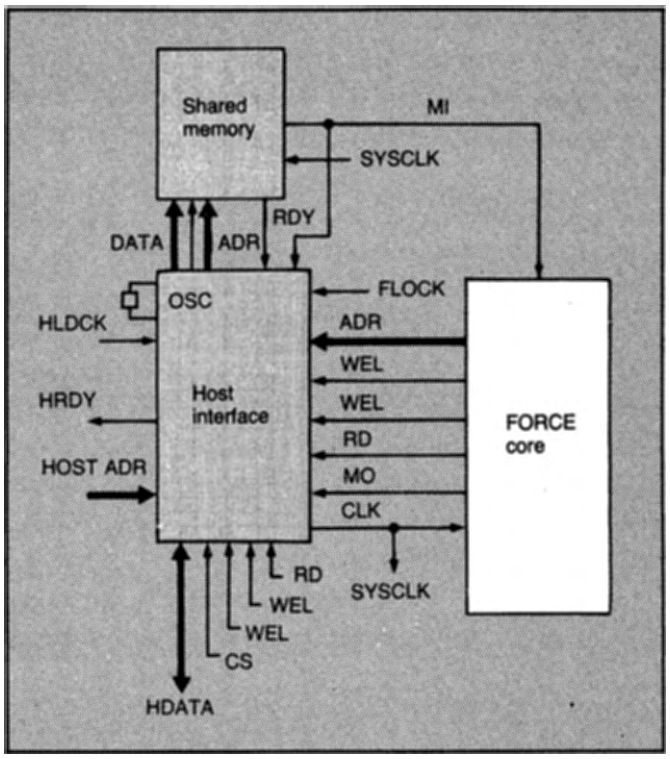


FIGURE 6. Configuration of core and shared memory with host interface.

nous host by synchronizing all commands from the host with the Force clock signal. The core processor can work with synchronous or asynchronous RAMs, even if the host interface is used in a completely synchronous system. If the interface is integrated with the core, minor modifications can improve response time in the host read/write cycle. Also, if the shared memory is integrated with the other components, the MEMORY_READY line is not necessary because the on-chip compiled RAM is fast enough to always contain valid data.

Multiplier

Harris has designed a 16 × 16-bit multiplier, using its proprietary MPS algorithm, for use with the Force core processor. The peripheral performs full 16 × 16 multiplication, generating a 32-bit product in as little as two clock cycles when the peripheral is integrated with the core; a breadboard system can complete a multiply in five clock cycles.

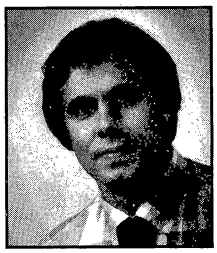
The multiplier operates either clocked or unclocked and provides tristate signals at the output control and data (MSP and LSP) lines. It has data latches at its inputs to allow clocked operation independent of the core. On-chip results are generated in less than 50 ns from the edge of the clocking signal, and the two 16-bit words in the product can be read simultaneously or in sequence.

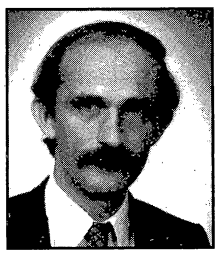
The designer can incorporate the multiplier into the Force architecture in several ways. First, he could designate several G-bus addresses to identify the multiplier, multiplicand, and the product's most-significant word (MSP) and least-significant word (LSP). Using this configuration, the core processor would write the addresses and read back the results to receive the result in four clock cycles.

He also could designate one G-bus address to identify either the multiplier or the multiplicand. The second operand can attach directly to the processor's TOP bus. To perform a multiplication, the multiplier would be written to the G-bus and the multiplicand placed on the TOP bus. Upon completion, the processor executes a FETCH_SWAP from the G-bus to receive either the MSP or LSP of the result (depending on the multiplier's configuration), placing it in the NEXT register. A G-bus FETCH then retrieves the remaining product word. This operation requires only three clock cycles to multiply two 16-bit numbers, and when many numbers need to be scaled by a constant (the multiplicand), the multiply takes only two clock cycles once the initial multiplier is written to the G-bus.

When the multiplier is integrated with the Force processor, the multiplier and multiplicand can be placed directly on the TOP and NEXT buses. The multiplier would be configured in its feedthrough mode, and a 32-bit product would be available every clock cycle. To execute a multiplication, the processor needs to read only the appropriate G-bus addresses. □

About the Authors





Peter S. Danile is currently a section head of semicustom design at Harris Semiconductor. Prior to joining Harris in 1981, he worked for both Northern Telecom and Motorola Communications. A graduate of the University of South Florida in 1976, Peter went on to receive his MSE with honors from Florida Atlantic University in 1980.

Christopher W. Malinowski is a senior scientist for Harris' semiconductor research and development department, and a program manager for the Force project. He holds an MS degree in nuclear electronics and a PhD in solid-state physics from Warsaw Technical University.

We're backing you up with products, support, and solutions!

SEMICUSTOM/CUSTOM
- CMOS Programmable Logic
- Gate Arrays
- Standard Cells
- Full Custom

TECHNOLOGIES
- CMOS Digital
- CMOS Analog
- Bipolar Analog
- Dielectric Isolation
- Gallium Arsenide
- Radiation Hardened

LINEAR
- Op Amps
- Comparators
- Analog Switches
- Buffers

DATA ACQUISITION
- Analog Multiplexers
- D/A Converters
- A/D Converters
- Sample-and-Hold Amplifiers

TELECOMMUNICATION
- SLICs
- PCM and Univ. Active Filters
- CVSDs
- T-1 and ISDN Circuits

DIGITAL COMMUNICATION
- CMOS 1553 Bus Interface
- CMOS UARTs
- CMOS Manchester Encoder/Decoder
- CMOS ARINC Bus Interface

MICROPROCESSOR
- CMOS 80C86—16-Bit
- CMOS 80C88—8/16-Bit
- CMOS 80C85 RH—8-Bit
- CMOS 80C86 RH—16-Bit
- CMOS 8/16-Bit Peripherals

MEMORY
- CMOS RAMs
- CMOS PROMs
- CMOS Memory Modules

GALLIUM ARSENIDE
- Microwave FETs
- Digital ICs
- Microwave Monolithic ICs
- Microwave Amplifiers
- Custom/Fabrication Services

RADIATION HARDENED
- SRAMs/PROMs
- Microprocessors
- Gate Arrays/Standard Cells
- Op Amps/Multiplexers
- Full Custom

Company Headquarters
2401 Palm Bay Road
Palm Bay, FL 32905
(305) 724-7418

International OEM Sales

Europe Headquarters (UK) 44-734-698-787
Japan (Tokyo) 81-3-345-8911

National Distributors

Anthem Electronics
Falcon Electronics
Hall-Mark Electronics
Hamilton-Avnet Corporation
R.C. Components
Schweber Electronics

In Canada
Hamilton/Avnet Corporation
Semad Electronics

FOR YOUR INFORMATION, OUR NAME IS

HARRIS

Harris Semiconductor: Analog - CMOS Digital
Gallium Arsenide - Semicustom - Custom

HARRIS

Reorder Number: 6AR-8019

HARRIS SEMICONDUCTOR REPRINT SERIES

Editorial Coverage by Leading Industry Publications

Standard-cell CPU toolkit crafts potent processors

Todd Jones, Christopher Malinowski, and Stanley Zepp
Harris Semiconductor Sector, P.O. Box 883, Melbourne, FL 32901; (305) 724-7000.

Reprinted with permission from Electronic Design (Vol. 35, No. 12) May 14, 1987,

A 16-bit, RISC-like CPU and complex support functions help a cell library called Force set the ideas of real-time designers into silicon.

Real-time-system designers demanding the highest performance are bound by hardware and software chains. System throughputs are shackled by the limitations of standard microprocessors, a poor match between popular languages and real-time control applications, and the lack of a powerful, general-purpose 16-bit microprocessor that can share one chip with application-specific logic.

Designers can overcome the speed limitations with bit-slice processors, which are hard to program; or by coupling custom logic and bipolar microcontrollers, which are power-hungry. Such solutions are expensive to develop and build, difficult to document and maintain, and often cannot be applied to new technologies.

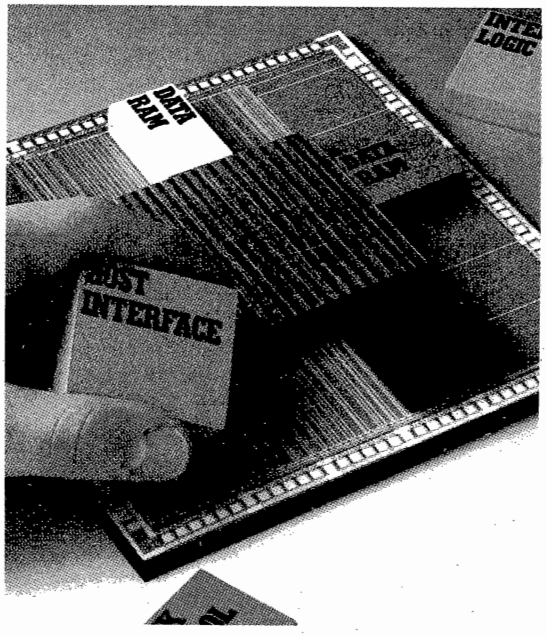
Out to break those chains is a Forth optimized reduced-instruction-set computing engine (Force) and a standard-cell toolkit. The CMOS engine is a 16-bit processor in standard-cell form, and the toolbox is a set of complex cells, among them a stack controller, interrupt controller, multiplier, and multiplier-accumulator. The cells fit into a computer-aided-design package for developing real-time products.

The architecture of the reduced-instruction-set processor puts to work in hardware an existing Forth-language virtual machine (see "Forth: A Language for Real-Time Control," p. 94). The silicon version grows out of technology licensed from Novix Inc., of Cupertino, Calif., and makes the virtual machine available as an embedded CPU and standard all-based product. The low gate count, however, does not sacrifice performance.

The RISC-like architecture avoids a high gate count, a problem that blocks other processors from serving as standard cells. The CPU has only 2500 gates, a very low number that owes to the processor's simple, directly executable set of Forth words and to heavy reliance on the two memory stacks. The CPU's low gate count leaves plenty of chip for stack memory, external interfaces, and specialized I/O to be created with cell-library CAD tools.

Depending on the task, the core processor can operate at clock rates in excess of 15 MHz, with an average execution throughput of between 1 and 1.5 clock cycles per instruction. That corresponds to a sustained throughput of 10 to 15 million high-level instructions per second (MIPS). Also, because each instruction executes the equivalent of several Forth primitives, peak processing throughput can exceed 30 million Forth primitives per second.

A typical form that the Force machine would take starts with the core processor and adds three memories: a main program memory, a return stack for dealing with subroutine calls, and a parameter stack for storing data (Fig. 1). Inside the core are three key registers that keep the operations moving at high speed. The I register, which is the logical top of the return stack; and the Top and Next registers,



which are the two top locations of a parameter stack.

Thanks to the small core size, designs can rely heavily on on-chip stack memories. For some tasks, data and program memories could also be included on chip. However, when large amounts of memory become too costly, fast, off-chip ROMs, EPROMs, and static RAMs enable designers to build systems operating at clock speeds of 10 MHz or greater.

HIGHLY PARALLEL ARCHITECTURE

All instructions execute in one or two clock cycles, and all three memory spaces can be accessed simultaneously. Although the concepts of RISC design apply to the processor, the architecture differs significantly from other CPUs of that type. The Force core puts to work a high-level language as its native instructions, giving the programmer a compact set of very powerful commands.

Other RISC processors use a reduced set of low-level instructions that help the chips optimize throughput. Often, however, programs are big and development time is long. In contrast, the Force core needs less code for a given application, increasing programmer productivity and cutting software-development costs.

The processor core executes instructions fetched out of main memory. These instructions closely mirror the Forth language primitives. Arithmetic and logic commands operate on the Top and Next registers and return the result to both those units, especially the Top register. Similarly, if operations must be performed on the return stack, the I register comes into play. There is also a fast I/O bus that can be accessed in parallel with the memories. For extra speed, arithmetic or logic operations, if needed, can be performed on read I/O data during, rather than after, the read cycle.

The main memory holds data, instructions, and the traditional Forth "dictionary" structures. Dual stacks are formed by two dedicated RAMs, which appear to the processor as last-in, first-out (LIFO) structures controlled by the stack-controller subsections (one for each stack). The stack controllers generate stack memory addresses under the direction of the processor.

In Forth, subroutine calls and operations on the data stack are most important. For this reason, the architecture is geared to these operations. For instance, the subroutine call takes one clock to do the "top-of-stack" arithmetic or logic operations. In addition, a subroutine return can occur in the same clock cycle as most other instructions. As a result, the total subroutine call-and-return overhead is cut to just one clock cycle with no extra time needed for the return.

The processor's highly parallel architecture executes the equivalent of several Forth language primitives at

Forth: a language for real-time control

Designing today's real-time control applications requires a high-level language that interfaces easily with custom hardware. The ideal language would need little or no dedicated memory outside that required for the application; it would present few, if any, restrictions on application-memory locations. The language must also lend itself to testing and debugging the application in its real-time environment. Finally, compiled application code should be compact and execute fast.

One language that easily meets all those conditions is Forth. No ancillary libraries or executives take up valuable memory space, and because Forth is quite compact, much of the development can actually take place on the application hardware. This ability greatly aids the integration and testing phase of the program. Run-time diagnostics are similarly aided by a small interpreter that does real-time monitoring and control in stand-alone tasks.

Forth is an integrated software-development environment incorporating an editor, compiler, and debugger, as well as a host of other development utilities that, in other environments, are usually separate. Because Forth is interpretive, the programmer can directly compile and execute code as soon as it is entered.

Being interpretive, the language speeds prototyping. It lets designers find out if the basic algorithm is correct before committing much time and resources to generating code.

Forth is a software environment that embodies a "virtual machine," which is the heart of the development system. Much of the virtual machine is the dictionary, which is a linked list of procedures called words. Stacks communicate parameters between procedures and link subroutine calls during execution; they are the hub of all machine activity. The parameter stack maintains the program data, and the return stack maintains the return addresses during execution.

A typical application is built upon subroutines, and each word that is defined in Forth is treated like a procedure call. An application is built up of words predefined by the programmer. These words can in turn be used again to define more complex words in the application. As each new word is defined, it is entered into the dictionary, from which it is pulled as needed. This process continues until the final application program exists as a single word.

The internal structure of a Forth word consists of a header, containing the name field and dictionary link, and the body. The body is a list of subroutine calls to the words that make up the definition. During execution of a word, the internal list of procedures is executed, calling another word of internal subroutine lists. This process of subroutine calls continues until a low-level primitive is encountered. That primitive is then executed, as it contains low-level machine instructions.

This process of layered subroutine calls is often referred to as threaded code. Because of this form of execution, the virtual machine depends heavily on modular programming techniques and subroutine calls.

ments, the cells can be cascaded using their Carry and Borrow signals. The cells generate the Stack Underflow and Stack Overflow outputs to indicate an empty and full status of the stack. Usually, these outputs would interrupt the processor.

For math intensive applications, the toolbox includes a 16-by-16-bit fast multiplier that executes a proprietary algorithm and operates at cycle times below 50 ns. The multiplier performs a signed-magnitude multiplication within one clock cycle of the engine. Also available is a 16-by-16-bit multiplier-accumulator (MAC) cell for jobs like digital filtering.

For dedicated tasks calling for parallel external ALU or concurrent external arithmetic operations, a set of fast 16- and 32-bit arithmetic cells is available. The library also includes a 32-bit barrel shifter and a leading-zero-detector cell for floating-point math operations that need fast normalization and denormalization.

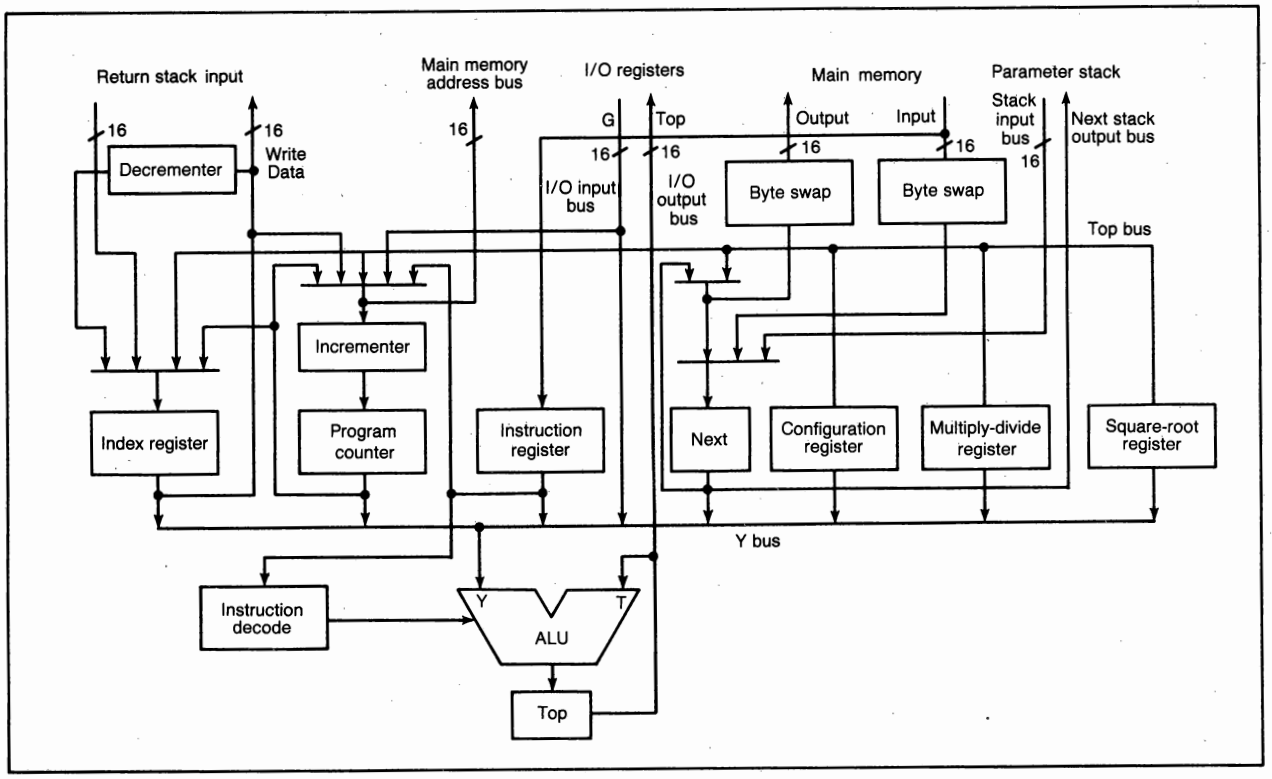
Giving priority encoding of up to 15 asynchronous maskable interrupts, the interrupt vector-generator (IVG) cell also delivers a 16th, nonmaskable interrupt (NMI) directly to the processor's NMI input. The IVG's mask register has a bit for each of the prioritized interrupt inputs and is loaded from the processor's T bus. That bus connects to the top of the parameter stack.

Any true interrupt input that is not masked will set the INT, informing the processor of the pending interrupt. At the same time, the IVG produces a vector location corresponding to the highest-priority unmasked interrupting input. This location establishes the point at which interrupt-code execution starts when the interrupt is acknowledged. The interrupt-vector location is read onto the core I/O-input (G) bus when the signal INTA emanates from the core during an interrupt-acknowledge cycle. That vector can also be read by subsequent I/O read instructions.

An asynchronous host-interface cell lets the core communicate with slower hosts, typically general-purpose processors, in a master-slave environment. In the interface scheme, the host processor can access either part or all of the chip's data or program RAM, which would appear to the host as a block of its own memory space. The interface cell's arbitration logic allows the host only one access at a time, interleaved with one or more core-processor accesses. This limit, however, may be bypassed with lock options by either processor. Those options grant temporary exclusive access to the interface by one processor or the other.

PUTTING THE PIECES TOGETHER

To see how all the macrocells come together, examine their work in a high-performance processor aimed at closed-loop control and other math-intensive tasks. Such jobs include robotics, instrumentation, flight control, sig-



2. A simple processor, the 16-bit Force core contains three key registers, the Top, Next, and I, that hold the most time-critical information. Other registers in the core take care of status-handling operations. Special registers and logic help multiply, divide, and find square roots.

once. In only two cycles, one such multifunction instruction performs the Forth equivalent of over swap—which exchanges, duplicates, and subtracts the values in two registers. It is executed just like classic executed Forth code.

The complex macrocells in the Force toolbox are based on an advanced cell library and computer-aided-design capability developed by Harris and SDA Systems Inc., of Santa Clara, Calif. The tools define cells and macrocells and efficiently place and route completed designs. They route designs incorporating fixed blocks of logic or memory, macrocells, and standard cells. Also included are tools to verify and simulate completed designs. The SDA software can compile RAMs and ROMs with variable size, configuration, and layout shape.

Besides the specially developed Force toolbox macrocells, the SDA design environment also contains a number of microprocessor-support peripherals, such as industry-standard and proprietary serial asynchronous transmitter-receivers, baud-rate generators, clock generators, programmable interval timers, and Manchester encoder-decoders. Not only that, the toolbox ties directly into Harris's already available standard-cell library. As a result, the designer can glue the complex blocks together or develop added logic functions using the 7400-series logic elements.

For breadboarding of systems, Harris has developed a 144-lead version of the Force core only. The core is included in a demonstration board from Logical Devices Inc., of Ft. Lauderdale. With this board, designers can access all of the processor's I/O, making it possible to breadboard a full system. It can interface to any CRT terminal or serial-communication port, and contains ROM and RAM space for the application code. Extra space is available should a particular task require more memory.

A development system configured as an IBM PC plug-in board and aimed at a PC-integrated development environment is now in final development by Silicon Composers Inc. of Palo Alto, Calif. Moreover, a Forth target compiler is now hosted on the IBM PC family. This target compiler makes possible the development of software in the PC environment, generating executable code for the Force processor and for use as a development tool for software vendors.

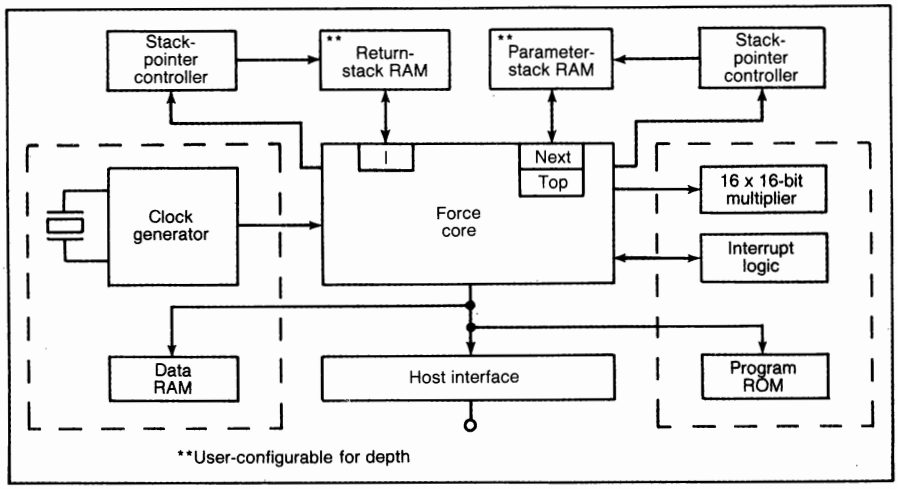
A CLOSE LOOK AT THE MACROCELLS

The Force processor core is the key element of the toolbox. That simple, yet powerful control engine is tied to other circuits by means of parallel, 16-bit data paths to the parameter stack, return stack, main memory, and the general-purpose I/O bus. There is also a 16-bit main memory address bus and a 5-bit address extension, which also functions as an I/O address bus. Inside the processor core are eight registers; four of which are independently accessible in parallel (Fig. 2).

The Top, Next, I, and Instruction Registers are all separately accessible so that multiple operations can be done simultaneously. The other four are the program counter, square-root, configuration, and multiply-divide registers. With its byte-swap logic on the main memory buses, the processor can rapidly reorder or perform byte reads or writes.

The core's two main work areas, the parameter-stack and return-stack memories, are addressed through identical stack-controller cells (Fig. 3), which generate address pointers within 5 ns from the time that the core's stack, read, and write signals become valid. The ability to quickly generate the stack addresses is critical to maximizing throughput.

The stack-controller cells can also be externally preloaded from the processor's data bus with a predetermined stack address. For deep stack-memory require-

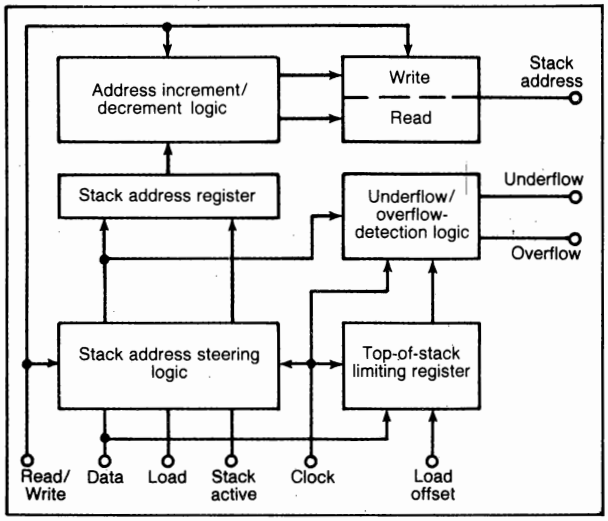


1. Harris Semiconductor's Forth-optimized, reduced-instruction-set computing engine (Force) core can be surrounded by the stacks and various support functions and memories to build a complete system on just one chip.

nal processing, graphics, and image processing. (That processor is, in fact, the first planned product to be built with the toolbox.)

The control processor takes advantage of the Force engine, the high-performance proprietary multiplier, and the normalize-shift macrocells (Fig. 4). It can function as a stand-alone processor but because it includes a host interface, it can share external main memory with any host processor. Provisions are also made to handle interrupts to and from the host processor.

Highly integrated, the chip includes two 128-word stack memories and controllers, one for the parameter stack and one for the return stack. The address registers of each of these macrocells can be loaded and read as I/O devices. On top of that, the overflow and underflow output lines from each of these macrocells drive interrupt inputs on the IVG.



3. One of the key support cells in the Force toolbox is the stack controller, which turns ordinary RAMs into last-in, first-out stacks for the processor. Two versions of the stack controller address 64 or 256 words of memory. Multiple controllers can be cascaded to handle larger memory spaces.

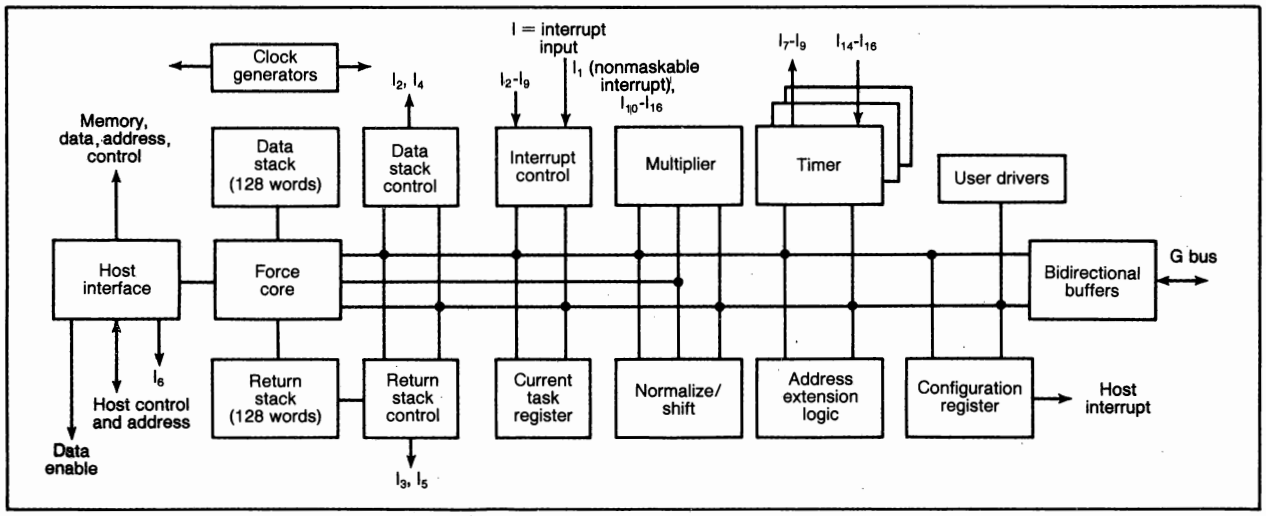
Arithmetic hardware sits on-chip to speed computations. That hardware includes a 16-by-16-bit multiplier and the normalize-denormalize-shift macrocell. The latter simplifies software development by delivering all the normalization that modern control systems typically demand for fast floating-point operations.

For control tasks, three 16-bit timer-counters on the chip supply a programmable time base, internal timing, or event-counting functions. These timers are clocked by a prescaled internal clock or by external inputs. A 16-input IVG macrocell obtains a fast response to internal or external events. The internal events flagged include overflow or underflow conditions for the four stacks and the three timeouts for the timers.

Provisions are also made for nine external interrupts including a host interrupt, a nonmaskable interrupt, and seven maskable interrupts. The interrupt mask register in the IVG cell can enable or disable any of these interrupts except the nonmaskable one.

The general-purpose coprocessor has the ability to address up to 16 Mbytes of external memory for code and data, all of which is external. Such a large range matches the addressing capability of most general-purpose host processors and supplies the space needed for software development, graphics, and image-processing jobs. It also makes possible complete flexibility with respect to memory configuration.

Because the core processor itself can produce only a 16-bit address (capable of addressing 64 kbytes of code or data memory), it is supplemented by memory-address-



4. A general-purpose coprocessor, with on-chip resources to handle fast integer multiplication and floating-point math, is easily assembled from the cells in the Force toolbox. Both the parameter and return stacks, as well as three timer-counters and an interrupt controller, are on the chip.

PRICE AND AVAILABILITY

The Force library is a part of standard product designs. The first such design is the coprocessor described in this article, which will be released in the fourth quarter. Prices for the coprocessor will be set then. It is also anticipated that the Force library will be released for semicustom design, but no date for that has yet been set.

CIRCLE 501

extension logic to achieve the 16-Mbyte range. Thanks to two 8-bit memory-extension registers within the address-extension logic, independent address extension is supplied for code and data to generate 24-bit addresses (16 Mbytes). Because the address extension for code and data is independent, maximum memory flexibility is gained.

An external processor works with a host-interface macrocell to gain access to the entire co-processor memory (16 Mbytes, if needed). This interface looks like a block of memory to the host, making it very easy to interface to any processor. The host interrupt to the Force processor is put into effect by a host write to a particular memory address. For stand-alone tasks, the host interface need not be used. Also included on the chip is a clock oscillator and clock generator, which supplies timing signals to the Force core, I/O devices, stack RAMs, and to the rest of the appliction system.

The core processor's I/O bus is brought off-chip to connect to specialized I/O devices required by the job. These I/O devices can also be built with the Harris—SDA standard cell gate-array design systems. For tasks that do not need all the internal I/O devices, an internal-configuration register selectively disables timers, math hardware, or address extension hardware, making their I/O addresses available to external devices.□

Todd Jones is a senior engineer at Harris, responsible for Force software planning and development. He has a BS degree in computer science from the University of Idaho and an MS in computer science from the Florida Institute of Technology.

Christopher Malinowski is a senior scientist for Harris's semiconductor research and development department, and program manager for the Force project. He holds an MS degree in nuclear electronics and a Ph.D. in solid-state physics from Warsaw Technical University.

Stanley Zepp, a senior scientist, is Harris's manager of business development for the microprocessor product line. He holds a BEE degree from the University of Florida and an MEE from New York University.

We're backing you up with products, support, and solutions!

SEMICUSTOM/CUSTOM
- CMOS Programmable Logic
- Gate Arrays
- Standard Cells
- Full Custom

TECHNOLOGIES
- CMOS Digital
- CMOS Analog
- Bipolar Analog
- Dielectric Isolation
- Gallium Arsenide
- Radiation Hardened

LINEAR
- Op Amps
- Comparators
- Analog Switches
- Buffers

DATA ACQUISITION
- Analog Multiplexers
- D/A Converters
- A/D Converters
- Sample-and-Hold Amplifiers

TELECOMMUNICATION
- SLICs
- PCM and Univ. Active Filters
- CVSDs
- T-1 and ISDN Circuits

DIGITAL COMMUNICATION
- CMOS 1553 Bus Interface
- CMOS UARTs
- CMOS Manchester Encoder/Decoder
- CMOS ARINC Bus Interface

MICROPROCESSOR
- CMOS 80C86—16-Bit
- CMOS 80C88—8/16-Bit
- CMOS 80C85 RH—8-Bit
- CMOS 80C86 RH—16-Bit
- CMOS 8/16-Bit Peripherals

MEMORY
- CMOS RAMs
- CMOS PROMs
- CMOS Memory Modules

GALLIUM ARSENIDE
- Microwave FETs
- Digital ICs
- Microwave Monolithic ICs
- Microwave Amplifiers
- Custom/Fabrication Services

RADIATION HARDENED
- SRAMs/PROMs
- Microprocessors
- Gate Arrays/Standard Cells
- Op Amps/Multiplexers
- Full Custom

Company Headquarters
2401 Palm Bay Road
Palm Bay, FL 32905
(305) 724-7418

International OEM Sales

Europe Headquarters (UK) 44-734-698-787
Japan (Tokyo) 81-3-345-8911

National Distributors

Anthem Electronics
Falcon Electronics
Hall-Mark Electronics
Hamilton-Avnet Corporation
R.C. Components
Schweber Electronics

In Canada
Hamilton/Avnet Corporation
Semad Electronics

FOR YOUR INFORMATION,
OUR NAME IS
HARRIS

Harris Semiconductor: Analog - CMOS Digital
Gallium Arsenide - Semicustom - Custom

HARRIS

Reorder Number: 6AR-8018

HARRIS

Standard Cell

September 1987

HSC 250 CMOS Cell Library

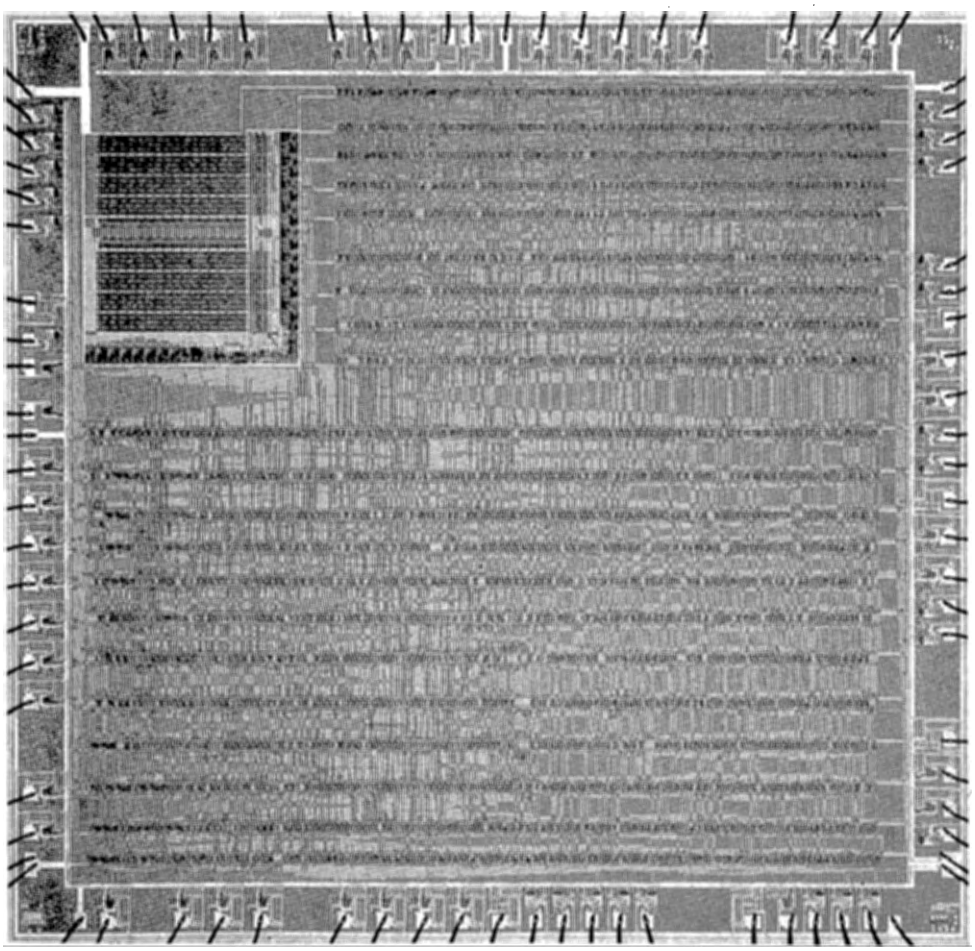
Features

- 1.5 Micron Effective Channel Length, 2-Layer Metal CMOS
- 1.2ns Typical Gate Delay Through 2-Input NAND
- Up to 100MHz Flip-Flop Toggle Rate
- Over 200 Primitive and Macrocell Functions
- Complex Function Megacells
- Customer Definable RAM and ROM
- Supported on Multiple CAE Platforms
- CMOS/TTL Compatible I/O's
- Commercial-Industrial-Military Temperature Ranges
- Proven Reliable and Manufacturable Process
- Extensive Range of Packaging Options
- Minimum 4kV ESD Protection
- Screening and Qualification to Mil-Std-883 Method 5004/5005, Class B
- Fully Compatible with the HSC200-RH Rad-Hard Library

Description

The HSC 250 STANDARD CELL LIBRARY is a proven, high performance dual-level metal library. The library offers a broad range of predesigned and fully characterized cells, macros, complex megacells and compilable RAM and ROM for developing reliable, cost effective customer specific IC's.

Die Photo



This information is current as of August 1987. Updates are issued semiannually.

Complex Function Megacells

To enhance the level of system integration, and reduce the design cycle time Harris has developed a series of complex function megacells. These functions consist of a family of highly integrated microprocessor peripherals, communication elements, high performance multipliers, and bit slice elements. A list of the available megacells follows:

• Microprocessor Peripherals

82C37A	DMA Controller
82C50A	Asynchronous Communication Element
82C50B	Asynchronous Communication Element
82C52	UART/BRG
82C54	Programmable Interval Timer
82C55A	Programmable Peripheral Interface
82C59A	Priority Interrupt Controller
82C84A	Clock Generator
82C88	Bus Controller

• Communication Elements

HD4702	Programmable Bit Rate Generator
HD6402	UART
HD6406	UART/BRG/Modem Control
HD6408	ASMA
HD6409	Manchester Encoder/Decoder
HD15530	Manchester Encoder/Decoder
HD15531	Programmable Manchester Encoder/Decoder

• Other Functions

H2901	4-Bit Slice ALU
*HMU16, HMU17, HMU18	16 x 16 Multipliers
**HMU1010	16 x 16 Multipliers/Accumulator

Compilable Cells

Harris has further expanded user definability by providing high performance, module compilation. This capability allows the customer to quickly generate design specific RAM and ROM cells.

RAM	Compilable to 16K
ROM	Compilable to 64K

*Contact Factory for availability
**Available Q1, CY'88

Absolute Maximum Ratings

Supply Voltage ..... -0.5V to 7.0V
Input/Output Voltage ..... VSS -0.5V
VCC +0.5V
Input Diode Current 10mA
VI < 0 or VI > VCC
Output Diode Current 10mA
VO < 0 or VO > VCC
Power Dissipation 1000mW
Continuous Supply Pin Current
VCC or GND 100mA
Storage Temperature
Plastic ..... -40°C to +125°C
Ceramic ..... -65°C to +150°C
Continuous Current per Output 10mA

CAUTION: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under "Recommended Operating Conditions" is not implied. Exposure to absolute maximum rated conditions for extended periods may effect device reliability.

NOTE: All applied voltages are with reference to GND (VSS).

Recommended Operating Conditions

D.C. Electrical Specifications VCC = 5V ± 10% T_A = Operating Temperature Range

SYMBOL	PARAMETER	MIN	MAX	UNIT	CONDITIONS
VCC	Operating Supply Voltage	4.5	5.5	V	
T_A	Operating Temperature Commercial Industrial Military	 0 -40 -55	 70 85 125	 °C °C °C	
VIH	Input High Voltage TTL CMOS	2.2 70% VCC		V	VCC = 5.5V
VIL	Input Low Voltage TTL CMOS		0.8 30% VCC	V	VCC = 4.5V
II	Input Current Standard Pull Up Pull Down	 -1.0 -500 -10	 +1.0 +10 +500	 µA	 VIN = VSS = 0.0V VIN = VCC = 5.5V
	Pull Up* Pull Down*	 +50	-50	µA µA	VI = 2.2V VCC = 5.5V VI = 0.8V
IOH	Output ~~Voltage~~ Current	6.0		mA	VOH = 2.4V; VCC = 4.5V
IOL	Output ~~Voltage~~ current		-6.0	mA	VOL = 0.4V; VCC = 4.5V
IOZ	Output Leakage	-10.0	+10.0	µA	VSS = VOL = 0.0V; VCC = VOH = 5.5V
ICCSB	Stand-By Supply		***	µA	II = 0; IO = 0
CI**	Input Capacitance	10.0 Typical		pF	VI = VCC or VSS; f = 1MHz
CO**	Output Capacitance	10.0 Typical		pF	VO = VCC or VSS; f = 1MHz
CIO**	Input/Output Capacitance	15.0 Typical		pF	VO = VCC or VSS; f = 1MHz

* Maximum input current for which specified VI will be maintained.
** Characterized at initial design and after any major design or process changes. Maximum values may vary by package type.
*** Customer design dependent.

Sales Offices

U.S. HEADQUARTERS
Harris Semiconductor
2401 Palm Bay Road
Palm Bay, Florida 32905
TEL: (305) 724-7418

EUROPEAN HEADQUARTERS
Harris/System Limited
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

DISTRIBUTORS IN U.S.A.
Anthem Electronics
Falcon Electronics
Hall-Mark Electronics
Hamilton/Avnet Corporation
Schweber Electronics

DISTRIBUTORS IN CANADA
Hamilton/Avnet Corporation
Semad Electronics

HARRIS

SEMICONDUCTOR PRODUCTS DIVISION

Reorder Number: 7DS-0097