Silicon compiler demands no hardware expertise to fashion custom chips

A functional language makes fast work of describing a custom processor. The compiler converts the description to an IC, and simulates its performance.

Instant expertise in VLSI chip fabrication is the promise of all silicon compilers. Since they automate most time-consuming and difficult steps of IC development, they are beginning to make custom VLSI and other application-specific ICs a feasible option for the system engineer.

But the level of input that each silicon compiler accepts varies widely. One type generates masks from logic schematics produced on a workstation. Another is satisfied with input at the architectural, or block-diagram, level. A third kind—exemplified by MetaSyn—works directly from a functional description of the chip and hence requires absolutely no hardware experience (Fig. 1).

Anyone with moderate programming ability can use this compiler. As the product evolves, even someone not involved in the original design can easily modify that source specification to include enhancements. Moreover, the compiler contains a high-level simulator that lets the designer observe the device's internal operation and its interaction with a simulated environment.

The compiler is based on the MacPitts silicon compiler, developed at the Lincoln Laboratory of the Massachusetts Institute of Technology. It permits the description of systems in algorithmic terms rather than in the structural terms of the hardware engineer. To clarify the difference between the alternatives, consider the following algorithmic fragment:

```
a := a + b − c
r := r −  a + d
```

In other words, first replace the value of a with

Jay R. Southard, MetaLogic Corp.

Jay R. Southard is vice president and director of technical marketing at Metalogic in Cambridge, Mass. After receiving an MSEE from Stanford University, he worked as a systems designer for General Instrument and Charles Stark Draper Laboratories. Most recently he was a researcher at MIT's Lincoln Laboratory, where he was active in the conception and implementation of the MacPitts language and compiler, which became the basis for MetaSyn.

the result of a+b−c, and then replace the value of r with r − a + d. This function can have many possible structural representations (Fig. 2).

Problem-oriented

The algorithmic approach is inherently less expensive to use than structural approaches. Thus it appeals to system designers who want to solve a system problem rather than create specific hardware; but it can upset hardware designers, because it does not permit them to specify and manipulate familiar hardware structures.

Theoretically the compiler cannot offer as great a variety of implementations as a hardware designer can. Practically, it solves nearly as many application problems, and of course, it does so in much less time and makes much more efficient use of silicon than do gate arrays and standard cells.

Although the MetaSyn specification for a chip is similar to a microprocessor program—especially for a bit-slice machine—it differs in several ways. Like a bit-slice microcode program, MetaSyn code may specify that several operations are to take place in the same clock cycle. The resulting parallelism clearly improves the algorithm's speed. Unlike microcode, however, a MetaSyn specification is not limited by some fixed, available hardware parallelism. Instead, the compiler automatically creates exactly the amount and kind of parallelism necessary to implement the designer's specification. This method also allows the algorithmic specification of such implementation techniques as pipelining.

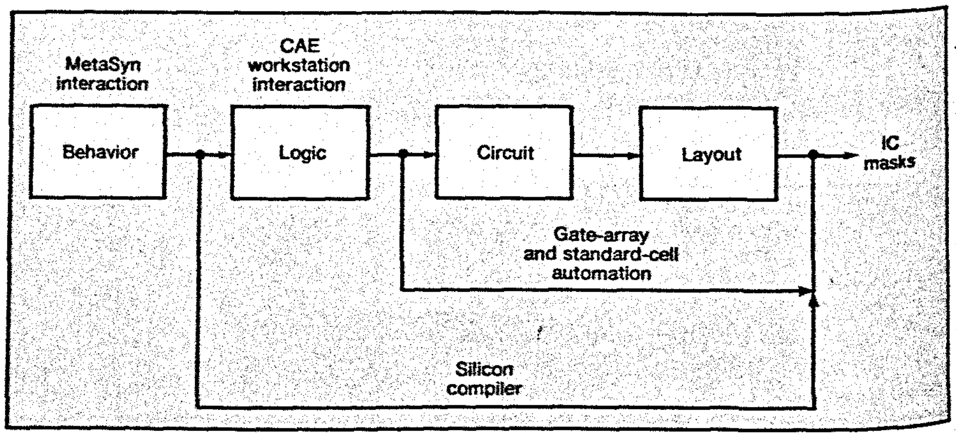
The new compiler goes a step beyond the way in which application programs are usually optimized. If a function's speed of execution is critical, it is usually coded in machine language. If it is even more critical and the computer has a writable control store, the function may be converted into microcode. The compiler goes beyond microcode. The application's critical algorithms need only be added to the processor's MetaSyn specification. The compiler then generates hardware that not only implements the old computer, but also the critical functions at a higher level of parallelism than available with microcode.

Two kinds of simulation

Before synthesizing the IC layout, the new compiler's high-level simulator mimics the device's behavioral specification. For this purpose, the compiler uses two kinds of simulation: interactive input with execution monitoring and system-level simulation.

The user interface of the compiler's simulator consists of a set of windows that monitor the high-level elements of the design: registers, processes, labeled instruction states, I/O ports, and other elements. Because these are also the elements of the compiler's behavioral specification, it is a simple matter for the designer to observe these elements and to use a mouse to modify their values (Fig. 3).

For simulation of a chip design—say, a pro-

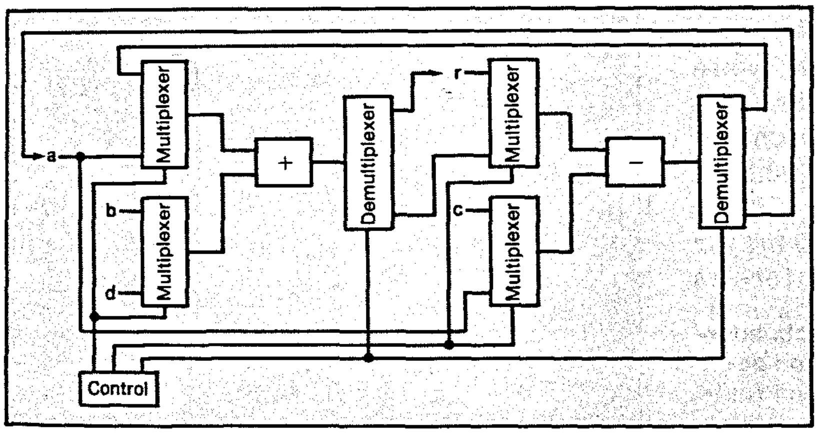


1. The MetaSyn silicon compiler starts chip design a stage earlier than other CAE software. It translates algorithmic descriptions of circuit behavior—not block diagrams—into logic hardware and ultimately into masks for fabricating ICs.

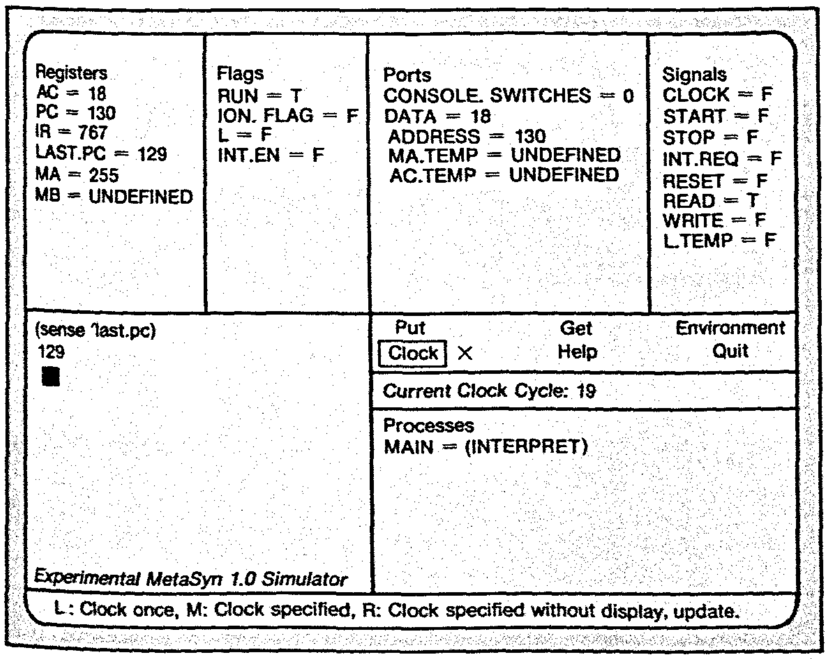
cessor—in the context of a complete system, the simulator creates several Lisp functions that can be used by other simulated system components to drive and sense the processor's ports and signals. This environment simulation thus deals with the same elements as the interactive simulator and the initial specification.

For example, the simulated processor can be connected to a simulated environment consisting largely of a memory that can contain a program for the simulated chip. The compiler simulates the processor and its environment concurrently, and the results can be monitored on the windows. Meanwhile interactive operation is simultaneously possible.

System-level simulation is also useful for control, signal-processing, and general system applications. In addition, the environment can



2. This circuit stores five values (a, b, c, d, and r) in master-slave registers and then funnels them through the adder-subtractors selected by multiplexers under the direction of the control box. It is not immediately obvious, however, that this layout is one of the many structural equivalents of the algorithm a:= a + b − c **and** r:= r − a + d.



```
Registers            Flags             Ports                          Signals
AC = 18              RUN = T           CONSOLE. SWITCHES = 0          CLOCK = F
PC = 130             ION. FLAG = F     DATA = 18                      START = F
IR = 767             L = F              ADDRESS = 130                 STOP = F
LAST.PC = 129        INT.EN = F         MA.TEMP = UNDEFINED           INT.REQ = F
MA = 255                                AC.TEMP = UNDEFINED           RESET = F
MB = UNDEFINED                                                        READ = T
                                                                      WRITE = F
                                                                      L.TEMP = F

(sense 'last.pc)                       Put            Get         Environment
129                                    Clock  X       Help        Quit
■                                      Current Clock Cycle: 19
                                       Processes
                                       MAIN = (INTERPRET)

Experimental MetaSyn 1.0 Simulator
 L: Clock once, M: Clock specified, R: Clock specified without display, update.
```

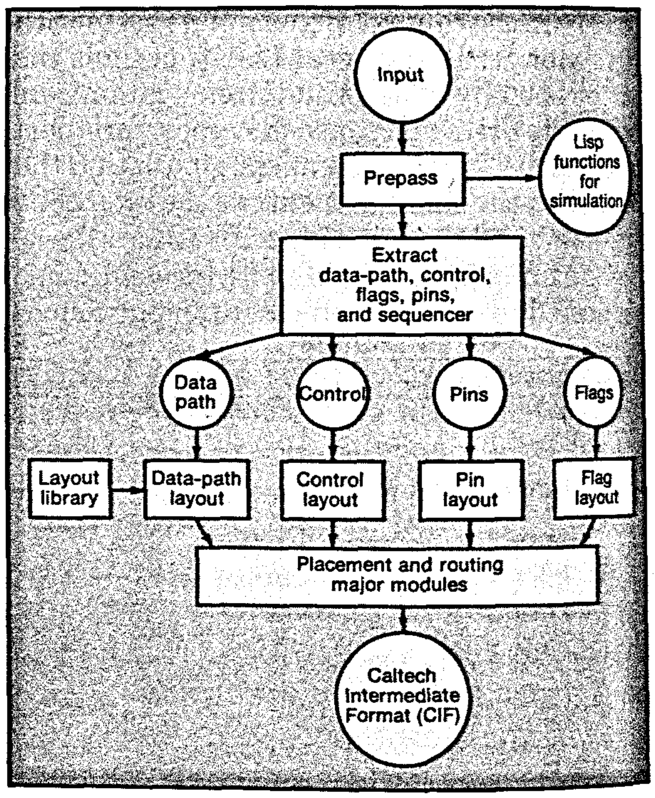
3. During simulation the register contents, flags, ports, and other internal functions of the designed chip can be followed on the screen. The effects of changes to the description are displayed within a few minutes at most.

produce a set of test vectors that mimic the environment, as well as the simulated chip, so that the chip, when fabricated, can be tested with standard automatic equipment.

Inside the compiler

A silicon compiler is a complex piece of software (Fig. 4). The input "source" description goes first through a prepass stage that checks for syntax errors, expands macros, and if the simulation option is in force, produces the Lisp functions for the simulator. Then the hardware components—registers, integer operators, logic gates, flags or pads—and their interconnections are "extracted" from the source specification.

The components are grouped into major modules: registers and integer operators as


Input
Prepass
Lisp functions for simulation
Extract data-path, control, flags, pins, and sequencer
Data path
Control
Pins
Flags
Layout library
Data-path layout
Control layout
Pin layout
Flag layout
Placement and routing major modules
Caltech Intermediate Format (CIF)

4. From the user's input, the compiler extracts information related first to simulation and then to specific processor functions. After placement and routing, it produces a CIF tape.

SEAGAT
DELIVE

part of the data-path module, logic gates as part of the control module, and so forth. Next each group of components is laid out with data-path, control, flag, and pad module generators. Since most of the interconnections are between the components of a module—between the registers and the operators of the data path, for example—much of the routing has now been done. Finally, the major modules are placed and routed to create the final layout in CIF (Caltech Intermediate Format).

Proof of the pudding

Over the past few years more than 50 MacPitts and MetaSyn examples have been generated, ranging from simple counters and shifters to signal-processing chips, computer peripheral controllers, and such microprocessors as an 8080 and a PDP-8. Even a neural network simulator has been built from MacPitts-generated chips. For simplicity, consider a stripped-down, 32-bit microprocessor called FRISC (Fanatically Reduced Instruction Set Computer) as an example.

The FRISC processor is based on a flexible and simple instruction set. It is specified in little more than three pages of text. However, as will be shown, the basic FRISC processor can be easily tailored to special-purpose applications and algorithms. When compiled, the FRISC specification produces the IC layout of Figure 5.

As seen in a pinout diagram (Fig. 5), the computer interfaces with its environment through a 32-bit bidirectional data bus and a 32-bit address bus. In addition, the microprocessor uses the Read and Write signal lines to control memory access. Interrupt Request and Interrupt Acknowledge lines handle interrupts, and a Reset line triggers power on reset. The computer is a stack-oriented machine with 4-bit-long instructions packed eight to a word.

The microprocessor contains five internal registers, although, as in any stack machine, a FRISC programmer will not be able to access

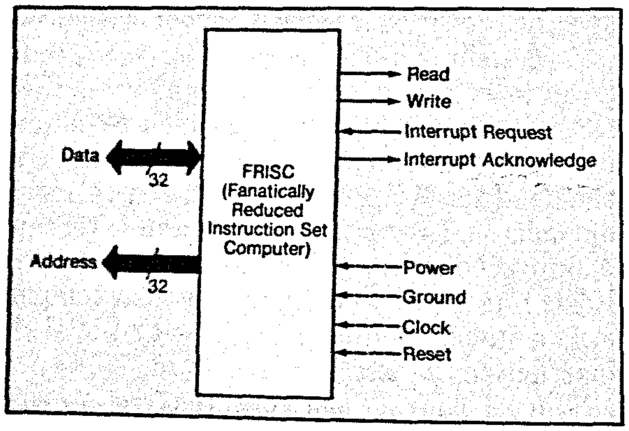
S NOW

NG

CIRCLE 97

them directly. Specifically p is a program counter; s, a stack pointer; a, the top-of-stack cache; b, the next-on-stack cache; and i, an instruction register. The core processor implements a few simple instructions (see the table, opposite). Because of the flexibility of the compiler, it is easy to expand, reduce, or modify this minimal set of instructions.

To implement FRISC, a straightforward microcodelike sequence will be used. The code will be broken up into reset, instruction-fetch, instruction-decode, and instruction-execution sections.



5. The 32-bit FRISC chip that MetaSyn will compile has two buses and eight other pins.

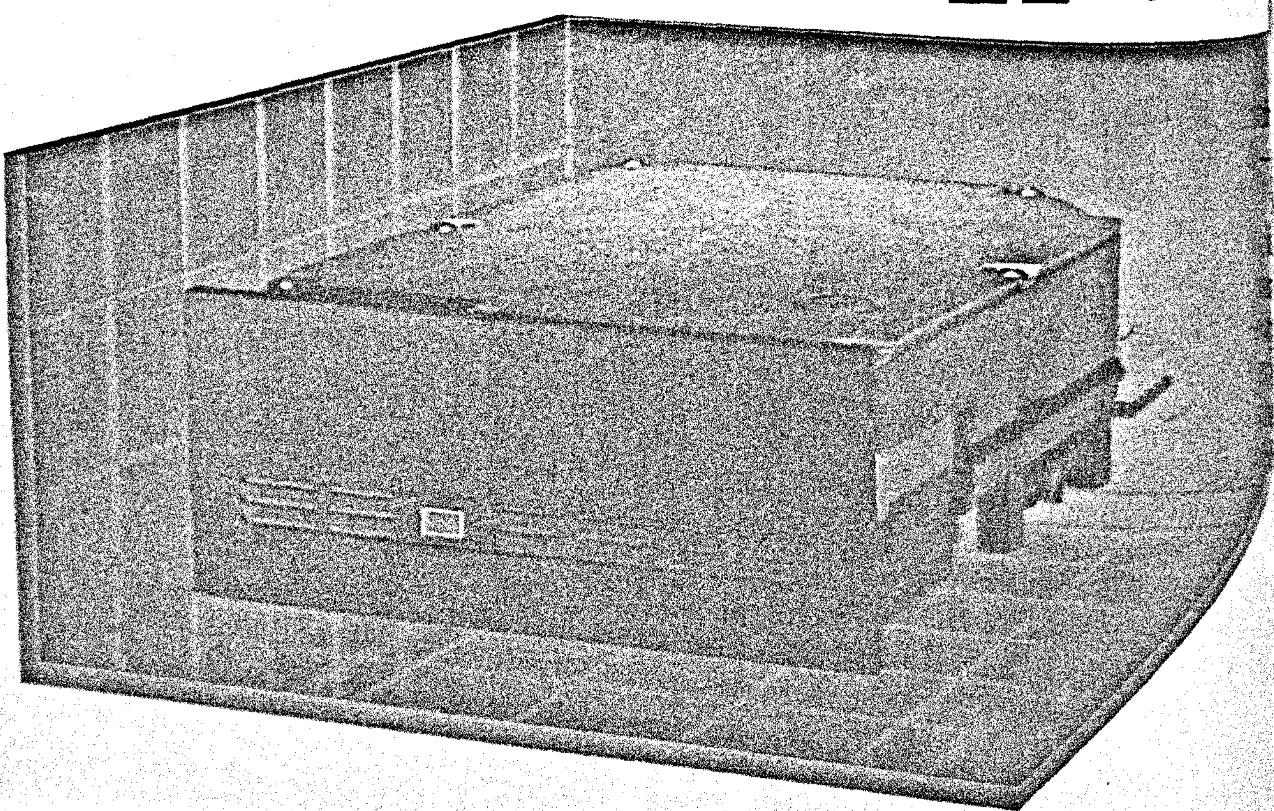
Back to state one

A MetaSyn machine with more than one instruction state must have a reset input signal that always returns the machine to its first instruction state. Thus Power On Reset (PRST) is the first state; it can be used to initialize the program counter (register p) with the data in memory location 0 and to initialize the stack pointer (register s) with the data in memory location 1. The state is defined as:

```
PRST
(par (setq address 0)(setq read t)(setq p data))
(par (setq address 1)(setq read t)(setq s data))
```

This code fragment specifies two clock cycles.

12MB IN



int
—
fo
an
po
(ty
sh
65
fo
le
pr
re

In the first cycle, the address port output is set to 0, the read signal is asserted (set to true), and the p register is set from the data port. In the second instruction cycle, the value in memory location 1 is accessed and read into register s. The MetaSyn par instruction specifies that all three setq clauses operate in parallel in the same clock cycle.

The following code segment checks for an interrupt request, and if none is pending, loads

FRISC instruction decoding

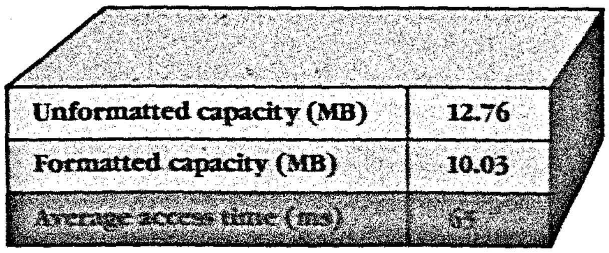
Instruction word*	Operation	Comment
0000000000000000	NOP	Instruction fetch
XXXXXXXXXXXX0001	ADD	Add top-of-stack elements
XXXXXXXXXXXX0010	INC	Increment top-of-stack
XXXXXXXXXXXX0011	PDSI	Push immediate data on stack
XXXXXXXXXXXX0100	LTM	Load from memory onto stack
XXXXXXXXXXXX0101	STM	Store into memory from stack
XXXXXXXXXXXX0110	SUB	Subtract top-of-stack elements
XXXXXXXXXXXX0111	SHFT	Shift top-of-stack right one bit
XXXXXXXXXXXX1000	IF	Conditional jump
XXXXXXXXXXXX1001	GO	Unconditional jump
XXXXXXXXXXXX1010	CALL	Subroutine call
XXXXXXXXXXXX1011	RET	Return from subroutine

* X = don't care
Bits 0 to 15 are not shown; their value equals that of bits 16 to 27

½ INCHES

Seagate downsizes the Winchester again. The company that introduced the first 5¼″ Winchester now offers a new low-cost standard—the 3½″ ST112.

It packs 12.76MB (unformatted) into the industry-standard footprint—just 4″ wide, 5.75″ long and 1.625″ high. It weighs only 2½ pounds, uses only 12 watts of power (typical), and withstands a 40G shock. Average access time is a fast 65msec. All of which makes it perfect for portables and desktops.



Unformatted capacity (MB)	12.76
Formatted capacity (MB)	10.03
Average access time (ms)	65

Seagate is committed to remaining the industry leader. We deliver more drives than anyone. We have the product choice, volume, quality, and price to meet your requirements. Call Seagate. You'll get immediate attention.

instruction register i, with the data in the memory location addressed by register p:

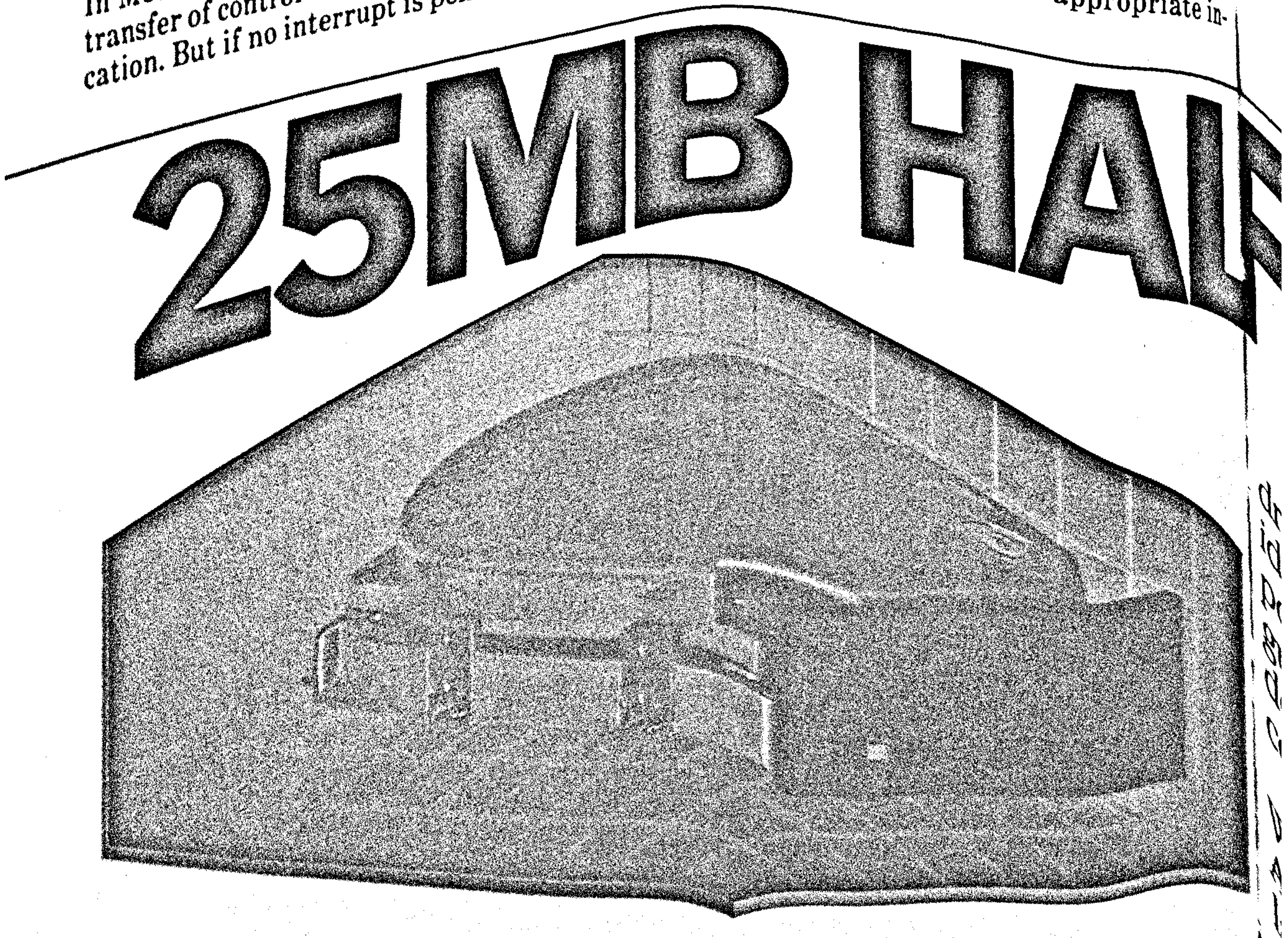
```
instruction-fetch
(cond   (interrupt-request    (go interrupt))
        (t                    (setq address p)
                              (setq read t)
                              (setq i data)
                              (setq p (1+ p))))
```

In MetaSyn parlance, cond resembles a case statement. Each branch of cond is a subexpression, guarded by the first expression within each of the example's two branches. The remaining expressions within the branch are executed in parallel, but only if the guard is true and previous guards are false. Therefore, the first instruction state of instruction-fetch, cond checks the Interrupt Request signal and, if it is true, aborts the instruction fetch and goes to the MetaSyn instruction state labeled interrupt. In MetaSyn, (go . . .) causes an unconditional transfer of control within the MetaSyn specification. But if no interrupt is pending, instruction register i is read in from the memory location pointed to by the program counter, register p. In a single clock cycle, the address port is set to p, the read control signal is asserted, and the i register is set from the data port.

In addition, this instruction state also contains the clause (setq p (1+ p)), which uses the built-in MetaSyn operator 1+ to increment the program counter. Because p is a register, it can be used as the source for address during the current instruction state (as required by the memory access) and still be incremented in parallel, because it will not change its value until the end of the current clock cycle. The compiler can be counted on to produce enough buses so that p can be routed to both the increment operator and the address port in parallel.

Case of the zero register

Now that an instruction resides in the i register, its four least significant bits must be decoded to pass control to the appropriate in-



struction execution subroutine (see the table, p. 193).

As the instruction in the four low-order bits is executed, it is shifted out of the i register. When that register equals zero, the current instruction word is exhausted, and a new one must be fetched. Thus the instruction-decode state must also check for the special case of the all-zero i register.

The instruction decode begins:

```
instruction-decode

(par (setq i (>>i 4 0))
     (cond ((=0 i)   (go instruction-fetch))
           ((eq? i 1 (3 2 1 0 ))  (call ADD))
           ((eq? i 2 (3 2 1 0 ))  (call INC)
```

Naturally there are more instructions, but these suffice to demonstrate the basic implementation. The code first shifts the i register, so that at the beginning of the next instruction state the register's content will be replaced by the same word, shifted to the right by four places and filled from the left with zeros. (The right-most position of these zeros is the least significant bit.)

While shifting the i register for the next clock cycle, FRISC simultaneously executes the condition or cond statement that will actually decode the current instruction. Each of the guards within cond conducts a check for a different op code, but the op codes are all checked simultaneously.

The first guard specified is for the all-zero instruction. The clause (=0 i) uses the built-in =0 function, which then tests its integer argument (in this case, i). A Boolean value of true is returned if the integer equals zero; any other integer yields a false. If the value is true, the matching go expression is executed.

If i is not zero, the next guard tests the four low-order instruction bits, as specified by:

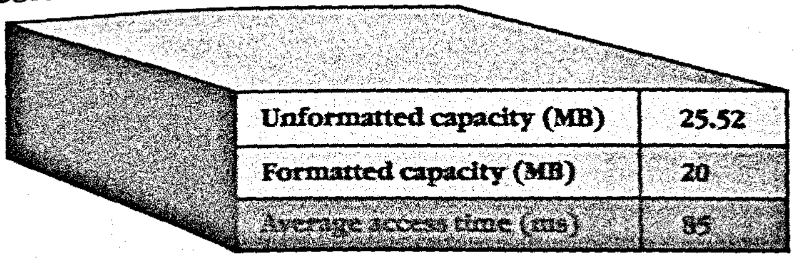
```
(eq? i 1 (3 2 1 0))
```

Again, eq? is a built-in function that takes a field of bits out of an integer (in this case, i) and

F -HEIGHTS

Seagate establishes the new standard in mid-capacity half-height drives—the ST225. Here is the 25.52MB (unformatted) capacity that's ideal for storage-intensive desktop systems and portables. With low power dissipation and high shock resistance. Plus proven technology guarantees immediate volume delivery and high quality at very competitive prices.

Nobody delivers like Seagate. We ship more drives than anyone. And every drive is backed by the largest, most responsive Winchester support team in the industry. We have the product choice, volume, quality, and price to meet your requirements. Call Seagate. And watch us respond.



Unformatted capacity (MB)	25.52
Formatted capacity (MB)	20
Average access time (ms)	85

compares the bits to another integer. A value of true results if the bits are equal and false if they are not. Thus this branch is executed only if the lowest four bits of i are equal to 1 (the op code for a FRISC ADD instruction).

If the branch is executed, control is transferred on the next clock cycle to the MetaSyn instruction state labeled ADD. Since this transfer comes about by means of a MetaSyn call instruction, the code at label ADD should end with a MetaSyn return instruction, in order to pass control to instruction-decode +1 (the next state after instruction-decode).

If the low-order four bits are not 0001 but 0010, then the current FRISC instruction is INC, and control is dispatched to that instruction state, and so on.

Ready for execution

It is important to realize that the compiler automatically generates the hardware to implement the specified parallelism; no further guidance from the designer is needed.

To see how FRISC executes an instruction, consider how, for example, increment is coded:

```
INC
(par   (setq a (1+ a))
       (return))
```

The functions par and 1+ are already familiar. In this case, the latter is called upon to operate on the top-of-stack cache, a. Incidentally, this top-of-stack increment and the program counter increment can physically share the same hardware, since the two increments occur during different instruction states—but the compiler will worry about all that. Simultaneously with the top-of-stack increment, control returns to the instruction-decode +1 state via the return operation.

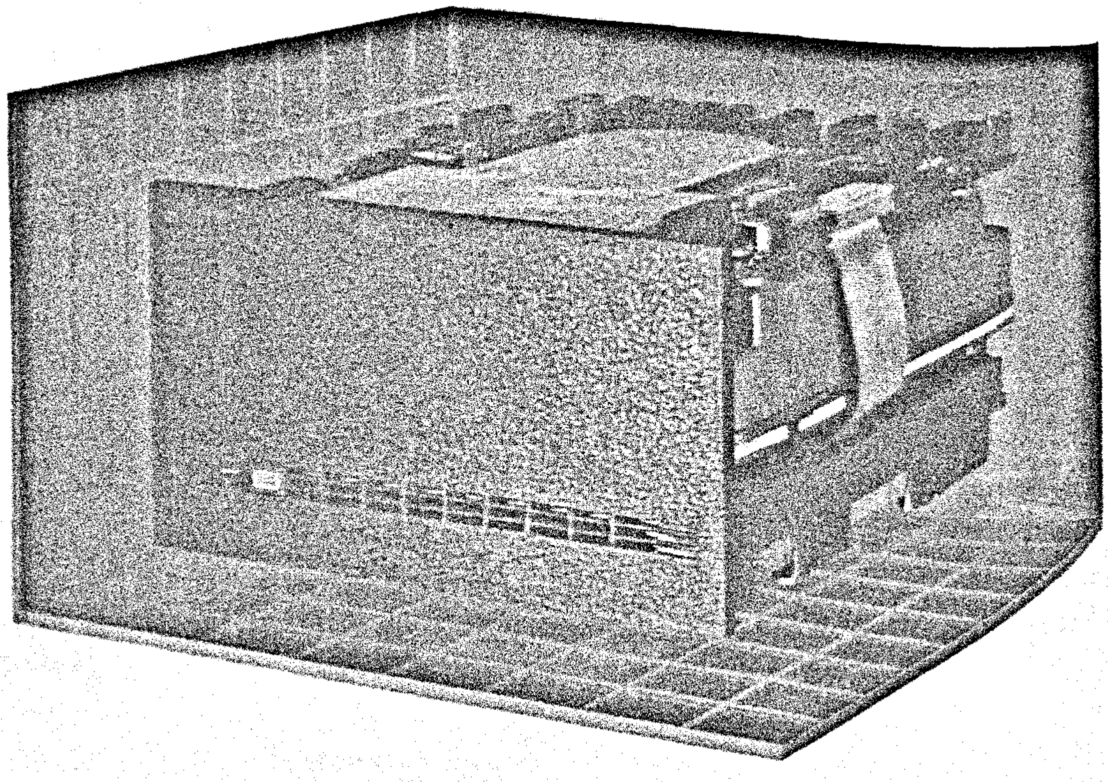
Once the specification for the computer has been completed, the designer should simulate it interactively by setting and observing the processor's I/O pins, or by running a FRISC pro-

gram i
to be u
tainin

In the

All
week,
pages
syste
factu
vital.
the fa
and c
single
the 1
out t
single
7 by
also
consu
Th
sisto

PERFORMANE



Se
time) ar
in volun
standarc
of multi
graphics
more cc
Featurin
and clos
Series h
proven,
Seaga
Winche
the pro
require

gram in a simulated memory. The test program to be used sums the elements of a vector containing the first six even integers.

In the chips

All told, the FRISC project took less than a week, including the specification of about three pages of MetaSyn code and two pages for the system simulation program. But the manufacturability and size of the FRISC chip is also vital. Of course, chip size is very dependent on the fabrication technology. A safe, inexpensive, and conservative technology would be a 4-μm single-level metal NMOS process. In this case, the 10,000-transistor, 32-bit FRISC chip turns out to be 7.7 by 9.3 mm. By going to a 3-μm, single-level metal process, the size gets closer to 7 by 6 mm—easily producible. The compiler also supplies statistics and estimated power consumption.

The chip's density, as measured in transistors per square millimeter, may appear low, but that parameter is nearly irrelevant. Far more important is functional density. For example, ROM, PLAs, and random logic are, for many purposes, logically interchangeable. The ROM implementation has the best transistor density, while random logic provides the same function with the fewest transistors. Very likely (depending on the function implemented), the PLA will emerge with the best functionality per square millimeter. Similarly, FRISC—or any other design—should be measured by functional density.

Build a better chip

One interesting aspect of a compiled chip like FRISC is the ease with which it can be converted into a 16-bit processor (or any multiple of 4 bits) by a change in just one number in the specification. For the 16-bit computer in the 4-μm single-level metal NMOS process, the size is 6.9 by 6.0 mm (Fig. 6).

Now that a FRISC prototype exists, it is time

E TO RELY ON

Seagate's new ST4000 Series offers high performance (40msec average access time) and a choice of capacities (26, 38, 51MB, unformatted) never before available in volume at such low cost. The ST4000 Series defines the new cost/performance standard in the industry. A standard that today is helping to make the new generation of multi-user, file server, networking, graphics, and CAD/CAM/CAE systems more cost effective than ever before. Featuring a linear voice coil actuator and closed loop servo, the ST4000 Series has been engineered with proven, manufacturable technology.

	ST4026	ST4038	ST4051
Unformatted capacity (MB)	25.62	38.17	50.88
Formatted capacity (MB)	20.15	30.02	40.02
Average access time (ms)	40	40	40

Seagate is changing the way the industry thinks about high performance Winchesters. We deliver more drives than anyone. We have the product choice, volume, quality, and price to meet your requirements. Call Seagate. You can rely on our performance.

to think about improvements. For example, it might be good to overlap the processes for fetching, decoding, and executing instructions, thus implementing a form of pipelining. Alternatively, an application might suggest an addition to the FRISC instruction set. Designers often do this in microcoded processors by adding the new instruction to the microcode; however, the new instruction can use only the existing hardware resources. But when a new instruction is coded in MetaSyn, it may result in new, more suitable hardware for the new instruction. For example, an instruction to sum the elements in a vector could be included. Such an instruction might be called vector-sum . It takes as the vector's base address the data in the top-of-stack element. The next element will contain the length of the vector.

When vector-sum is completed, these two values are popped off the stack, and the sum is pushed on top of the stack. During execution of vector-sum, the a register serves as the current address in the vector, and b holds the vector's remaining count. No register is needed to hold each element as it comes in from memory, since FRISC can be made to add directly from the data port. To keep the running sum, however, a new register, called vs (for vector sum), becomes necessary.

Just add nine lines

To implement the vector-sum instruction, FRISC's instruction-decode section must be modified and new code added to the instruction-execution section. One way to implement the latter is:

```
vector-sum
   (cond (( = 0 b) (setq b vs)
                   (go pop))
         (t  (setq b (1-b))
             (setq address a)
             (setq read t)
             (setq vs (+ vs data))
             (setq a (1+ a))
             (go vector-sum)))
```

NOBODY D
SEAGATE

The MetaSyn code for the instruction consists of a single instruction state. First the code checks for the end-of-loop condition—that is, whether all elements of the vector have been summed. This is done by means of an = 0 operator, as previously used in the instruction-decode section. If the summing has been completed, the result is placed on the stack, and the stack is cleaned up—in other words, the vector's start and length value are removed by a jump to the pop instruction code. Since that code ends with return, control returns to instruction-decode + 1.

If the sum is not completed, the memory location addressed by the a register is read while the current value of the vector length, stored in the b register, is simultaneously decremented via the built-in MetaSyn operator 1−. Contrary to what happens in a normal reading operation, the actual data from the memory is not stored, but is added to the value in vs, the running sum register. At the same time the current vector element address (in register a) is incremented for the next time through the loop. Again, because of the implied register timing, the value in the register will not change until the beginning of the next instruction state, so that address is maintained constant during the memory reading process. The final expression in this branch of cond sends control back to the vector-sum label for the next instruction state.

A good trade

When compiled, the 16-bit computer chip, complete with vector summing, measures 7.5 by 6.2 mm in area, or 12% larger than the original 16-bit FRISC. In return, it executes a vector sum more than 10 times faster than the old chip could—with software alone.

The redesign of FRISC involves one other cost—namely, the additional design time. In this case, the modification is so simple that it takes about 2 man-hours to modify the FRISC MetaSyn specification, create a vector-sum test

IVERS LIKE

CHOICE • VOLUME • QUALITY • PRICE

Regional Sales Offices:
San Jose, California (408) 286-7580;
St. Petersburg, Florida (813) 577-1199;
Natick, Massachusetts (617) 486-9711;
Dallas, Texas (214) 783-6711.

Canada:
Semad, (416) 475-3922; TWX 6104924455

European Sales Office:
49 89 177017, TELEX 524275 SEAG D
Seagate is a trademark of Seagate Technology
©1984 Seagate Technology

920 Disc Drive,
Scotts Valley, California 95066
Telephone (408) 438-6550
TELEX 176455SEAGATESCVL

Authorized U.S. Distributor:
Hamilton/Avnet

SEE US AT COMDEX, BOOTH 4305

Seagate

program, and test the design using the MetaSyn simulator. Two more hours of compilation time is needed to produce the layout.

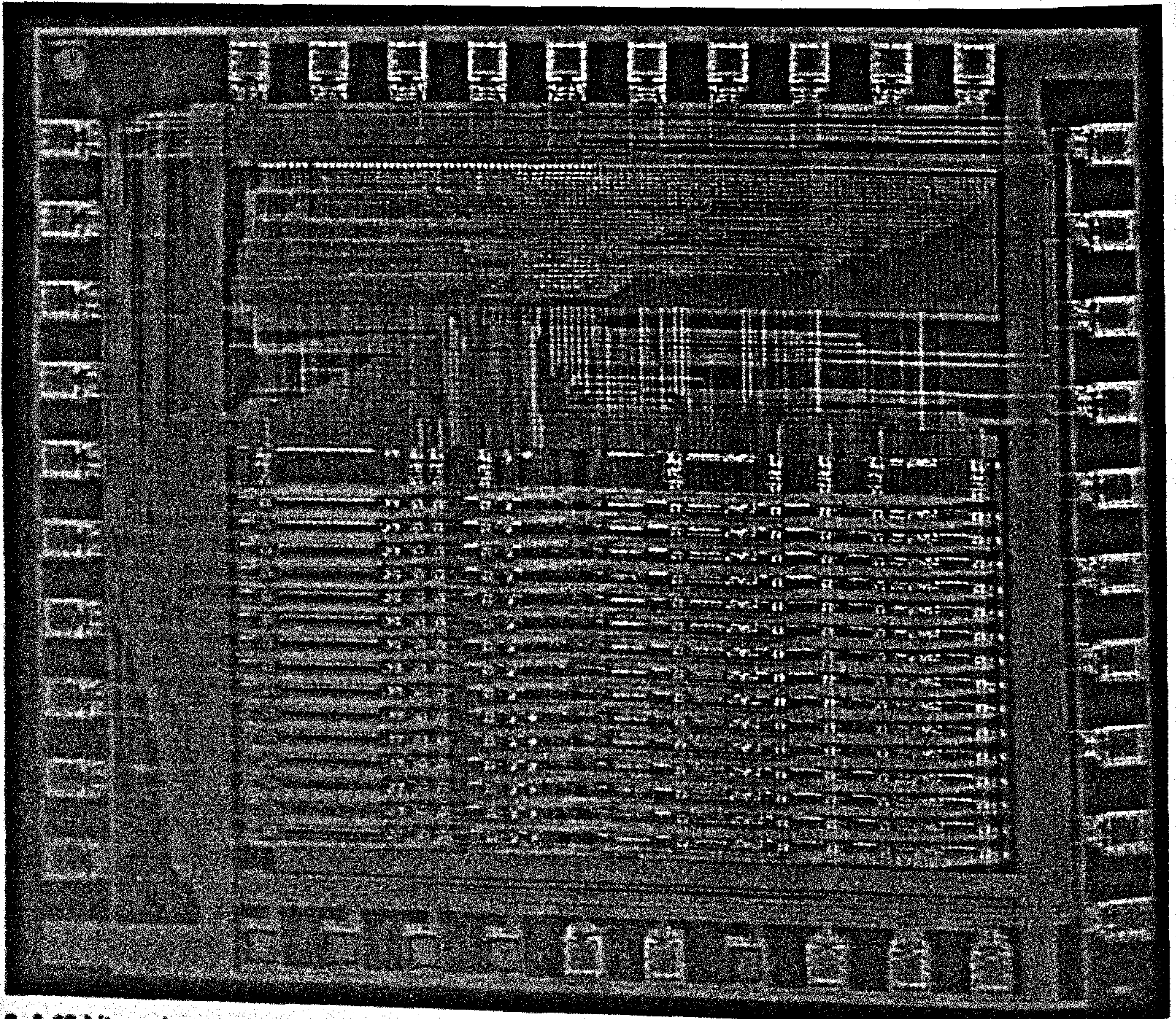
Speedy compilation

Although the overall design time is most important, the compiler's actual CPU time is also of interest. There are two kinds of compilation time: the time from specification to interaction with the simulator, called "compilation to simulation," and the time to create the layout.

Compilation to simulation is very important, since it represents the innermost design loop. The largest designs compile to simulation in a few minutes, and the interactive response times are hardly affected by the size of the design.

As to layout time, small designs have compiled in a few minutes to half an hour; the largest designs, such as the FRISC chip, can take from two to four hours. These times apply to the August 1984 version of the MetaSyn compiler, running on a Symbolics 3600 Lisp machine with 474 Mbytes of disk storage and 1 Mword of semiconductor RAM.□

How useful?	**Circle**
Immediate design application	547
Within the next year	548
Not applicable	549



6. A 16-bit version of the computer chip, ready for conversion to the CIF pattern-generation tape, has dimensions of 6.9 by 6 mm, based on a 4-μm single-metal NMOS process.