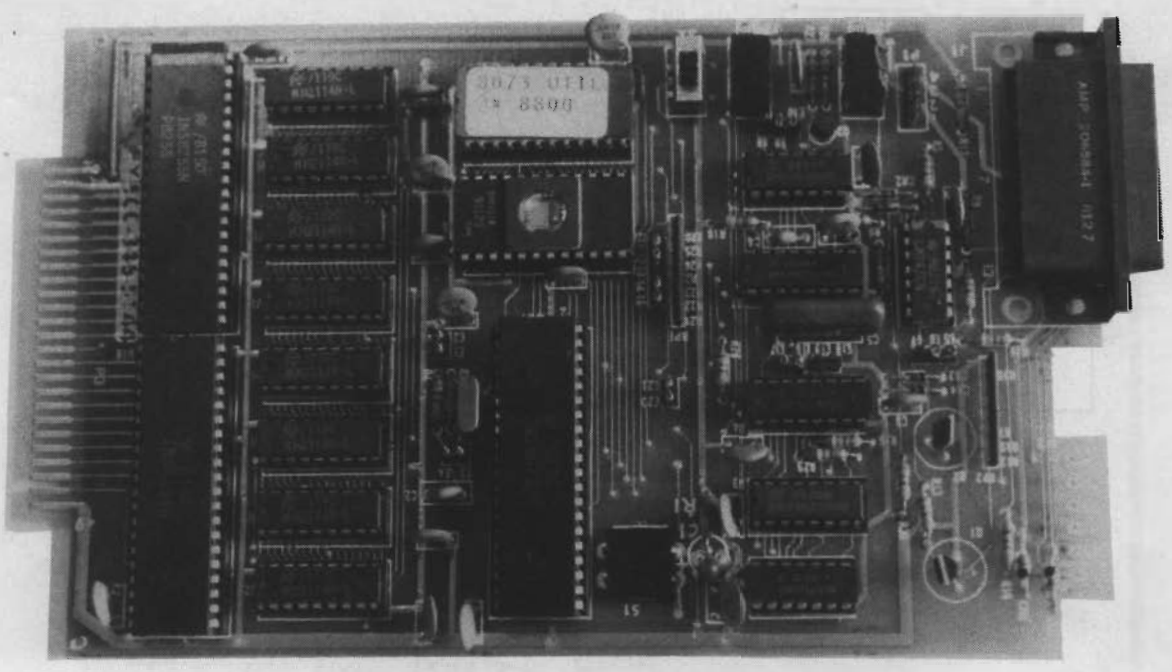

How to Design Microprocessor-based Projects

Make your next project "intelligent." Use a single-chip microcomputer to control it.

TOM FOX

ARE YOU FASCINATED BY MICROprocessor-controlled, semi-intelligent gadgets like robots that avoid obstacles, microwave ovens that monitor the food as it cooks, and cars that adjust the engine so that it's working as efficiently as possible? Have you come up with hundreds of control-computer applications that you'd like to try? (But you can't picture tying up your powerful personal computer for control applications?)

There *is* a way to design computer-controlled "smart" devices without tying up your expensive personal computer. It's easy if you use a device called a *microinterpreter*.

In this article, we'll show you how to design almost any computer-controlled device your imagination can dream up—even if you've never worked with microprocessors before! That's because you don't have to learn machine or assembly language and you don't have to invest gobs of money in expensive and complex development systems in order to design a computer-controlled machine. As long as you have some knowledge of electricity and digital logic circuits you should be on your way. But some familiarity with the BASIC programming language, as well as binary and hexadecimal numbers will come in handy too.

To get you on your way toward designing practical, semi-intelligent machines, we will describe, in detail, the design of a such a device: a "burglar outwitter." As its name implies, the purpose of the outwitter is to fool a potential burglar into thinking someone (or something) is home when actually no one is there.

What's a microinterpreter?

If you have ever used today's microcomputers, you probably realize that they're fairly easy to use. However, microprocessors—the brains of the microcomputer—have a notorious confusing nature about them. What is it that turns a complex, often mind boggling, microprocessor into an easy-to-use microcomputer? *Software!* Although it seems at times you can do magic with it, there is nothing magical about software. It's merely a list of detailed instructions that tell the microprocessor exactly what to do.

That's the same thing that makes a microinterpreter easier to use than a microprocessor. A microinterpreter is basically a single IC that contains both a microprocessor and software. The software in microinterpreters is contained in ROM (*R*ead-*O*nly *M*emory).

Before we go any further, we should point out that microprocessors with a built in high-level language are not universally called "microinterpreters." A few semiconductor companies refer to those IC's as single-chip microcomputers. However, that name is not only clumsy, it is unnecessarily vague since many of those same companies call microprocessors that have no built in high-level language "single-chip microcomputers." National Semiconductor refers to their INS8073—the device that we'll examine in detail—as a microinterpreter. This seems to be the best name available, and we will stick to it.

You might be thinking that there must be a catch somewhere. Microinterpreters

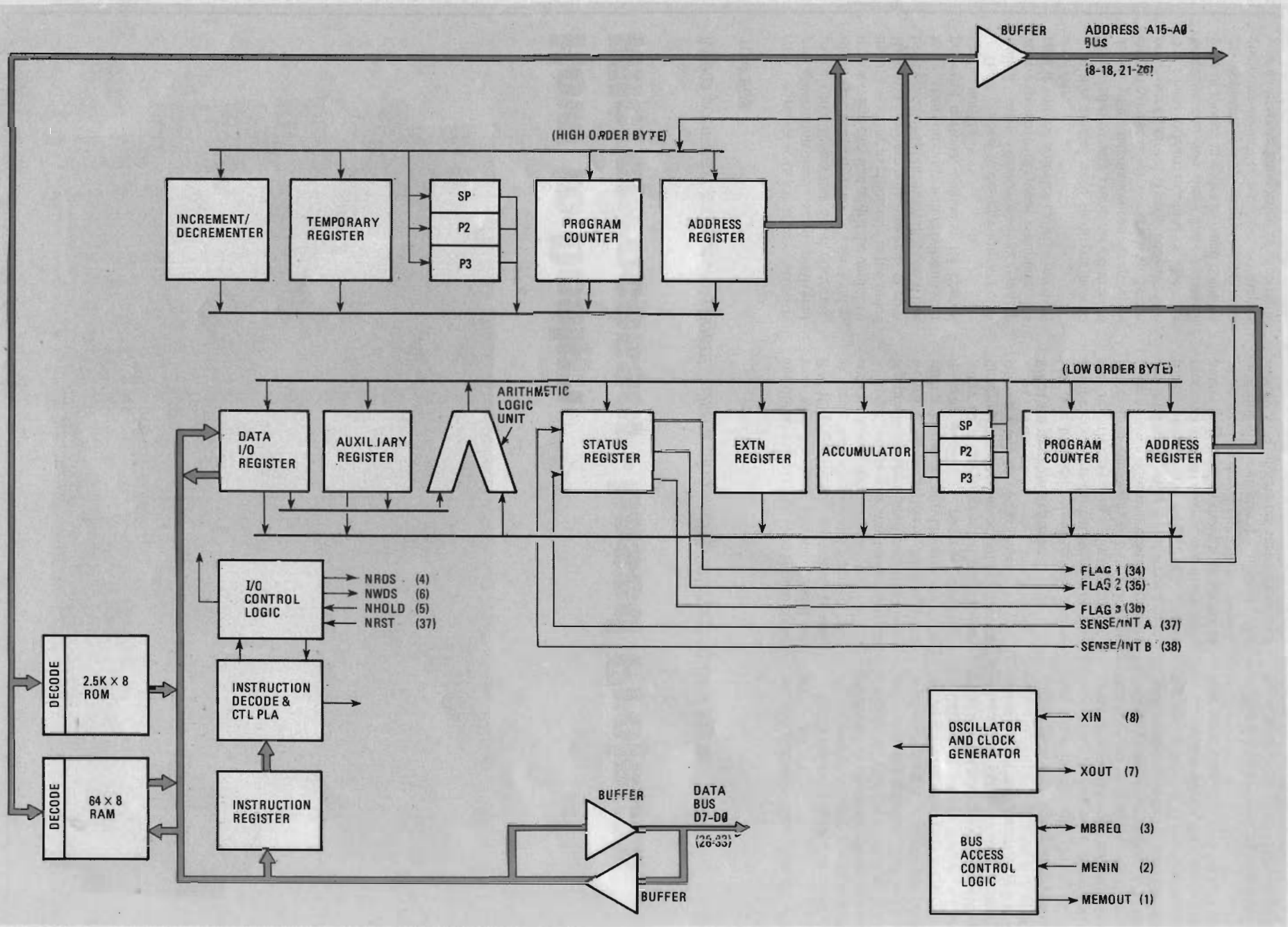

FIG. 1—BLOCK DIAGRAM OF THE INS8073 from National Semiconductor. Note that the microinterpreter contains 2.5K of ROM and 64 bytes of RAM.

seem almost too good; they must have a few bad points. Well in certain applications, they do. For example, if you're planning to design a revolutionary new personal computer that will take the peel right off the Apple, you probably should look past microinterpreters—they are relatively slow. That's because the interpreter (located in ROM) must convert the simple English-like commands into complicated machine language. And it must do that line by line as the program is running. That process takes a bit of time.

The good news is that the microprocessors in microinterpreters are so fast that for most control or monitor applications, they're much faster than the situation requires. But in those applications where a jiffy is several microseconds too long, you can program the time-consuming subroutines in assembly language. But that's enough talking in generalities—let's look at a specific device: the INS8073. It's block diagram is shown in Fig. 1.

The INS8073 microinterpreter

As we mentioned earlier, a microinterpreter is basically a microprocessor that understands a high-level language. Such microinterpreters fall into two general categories: ones that understand Tiny BASIC and ones that understand a form of stripped-down Forth.

Forth was developed in the early 1970's for real-time control of astronomy equipment. Because of that, it is an excellent language for microinterpreters since one of their primary uses is in control applications. Forth's main advantage, compared to Tiny BASIC, is that it is faster. However, unless you have already programmed in Forth, you'll have to spend a considerable amount of time learning the language. While Forth enthusiasts might write some letters to the contrary, most people would agree that Tiny BASIC is easier to learn than Forth.

Even if you have never programmed a computer before, you'll be able to write a program that works using BASIC in less time than with just about any computer language there is. It's true that many BASIC programs might not be very elegant. Nonetheless, if the program does what we want it to, then it's just fine. After all, that's what we're after—to get the job done.

Another advantage of BASIC is that it is the most widely used language among microcomputer users. (*Apple II, Timex, Sinclair 1000, Commodore 64* and many other popular computers all have a form of BASIC built-in.) So there is an excellent chance that you are at least mildly acquainted with BASIC. And if you are, your job in learning how to design smart machines with microinterpreters is nearly half done.

In our quest for simplicity, we will examine only one microinterpreter: the INS8073. However, you will be able to apply much of what we say to other devices. That IC, though relatively simple in architecture, has noteworthy features like 16-bit hardware multiply and divide. However, the best thing about the 8073 is that its language (National Semiconductor's Tiny BASIC) is a true gem. While we will describe that language in more detail later in this article, a few highlights will be given here.

National Semiconductor's Tiny BASIC includes DO-UNTIL commands. That type of loop control was borrowed from PASCAL and is unusual for Tiny BASICS. The ON command simplifies the handling of interrupts, which is important when designing smart machines. Another convenient instruction is DELAY, which can be used to pause program execution for a user-determined length of time. This BASIC includes a RND function, which we will use extensively when we design our burglar outwitter.

National Semiconductor's Tiny BASIC handles strings and uses the operator (which simplifies transferring data back and forth between the program and memory locations or input/output ports). There are other unusual features of a Tiny BASIC, which we'll discuss in more detail later. Actually, if it were not for its restriction to integer numbers and 26 variable names, National Semiconductor could leave out the adjective "tiny' from the language's name and hear few complaints.

The 8073 has already been chosen by designers for a number of commercial applications. For instance, it is the "brain" of RB Robot Corp's *RB5X* robot and has been used for precision measurement of conditions in oil wells and testing the feasibility of the digital design of FM tuners.

Inside the INS8073

The INS8073 microinterpreter is an INS8072 8-bit microprocessor that contains Tiny BASIC in its 2.5K on-chip ROM. It has a 16-bit address bus and requires a single 5-volt supply voltage. Other features of interest to us include an on-board clock, TTL compatibility and 64 bytes of on-board RAM. (Yes, more RAM would be nice.)

There are some attributes of the 8070 series of microprocessor that affect us only indirectly. Those include 8 *and* 16-bit arithmetic, logic and stack-manipulation instructions, hardware multiply and divide, and single-instruction ASCII-to-decimal conversion. The 8072/8073's multiprocessing feature will not be used.

A feature of the IC that simplifies its use as a stand-alone, real-time controller, is the provision for automatic execution of ROM (or EPROM) at power-up or on reset. The 8073 also contains firmware that allows easy interfacing to a RS-232 terminal.

The INS8073 "understands" ASCII (the *A*merican *S*andard *C*ode for *I*nformation *I*nterchange) so programs can be simply and quickly modified without the need for costly and awkward assemblers, text editors, debuggers and development systems—all you really require is a terminal (or a computer and terminal software) to enter a program into the microinterpreter's RAM.

Figure 1 is a block diagram that shows the architecture of the INS8070. Since this article is primarily concerned with the design of microprocessor-controlled equipment using only Tiny BASIC, we will ignore the fact that the 8073 has a machine level instruction set. However, if we are to design real working circuits, we must examine the IC's pin configuration as well as the function of each pin in some detail.

Pin functions of the 8073

Figure 2 shows the pinout of the INS8073. While we could go through the pins, starting at pin 1 and explaining each pin's function in sequential order, we will start at the easy part first—the power supply connections.

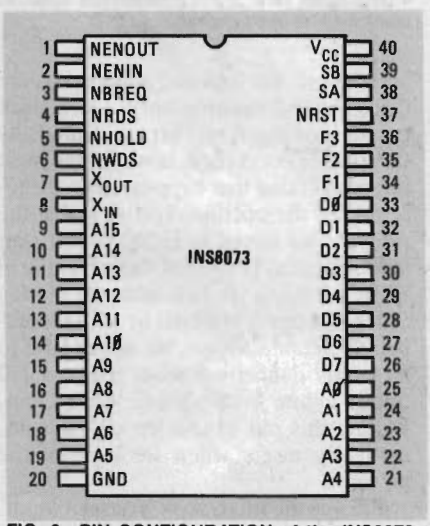

FIG. 2—PIN CONFIGURATION of the INS8073.

Pin 20 is the GROUND pin. It should always be connected to the logic power supply's ground. Pin 40 must be connected to a well-regulated voltage source (+5 volts with respect to pin 20). That voltage shouldn't vary by more than ± 5% (so you have a permissible range from 4.75 to 5.25 volts). The power supply should be able to supply a minimum of 250 ma.

Pins 26–33 are the connections for the bidirectional 8-bit data bus. (The most significant bit, D7, is connected to pin 26.) The data bus is three-state, which in this case means that it is basically disconnected from the circuit except for external write operations. The data bus can drive one standard TTL load or several LS-TTL loads. Pins 9–19 and 21–25 are the address-bus pins. That is also a three-state bus.

In order to use the on-board oscillator, an external crystal must be connected to pins 7 and 8. Figure 3 shows how. If you prefer to drive the IC with an external clock (it must be TTL compatible), connect the signal to pin 8. Pin 7 provides a buffered clock output with either an external or internal clock.

Pin 37 is the RESET pin. It is simple in concept, but extremely important. When it is brought low (below 1 volt) several things happen. Any in-process operations

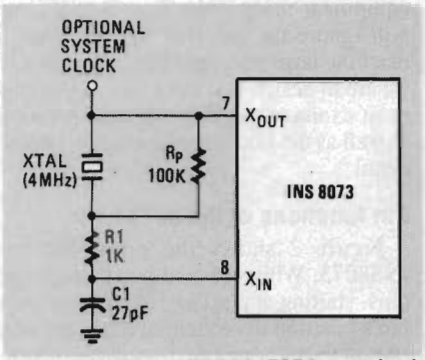

FIG. 3—A INTERNAL OSCILLATOR is contained in the INS8073. Here it is shown configured for a 4-MHz clock rate. A TTL-compatible external clock may be used instead.

are aborted, the data and address bus are disconnected, the program counter, stack pointer, and status register are cleared. As far as we're concerned, however, the most important thing that happens upon the resetting of the microinterpreter is that the program we stored in EPROM will start automatically. (Provided the program in EPROM starts at hex address 8000.) Since that pin is buffered by a TTL compatible Schmitt trigger, we do not have to be overly concerned about rise and fall times. Figure 4 shows one way of connecting this pin so that the device automatically resets when we turn on the power.

Pin 4 is the READ DATA STROBE output. It is a three-state output that goes low when the microinterpreter is reading from external memory.

Other than those times, the output is effectively disconnected from the rest of the circuit. In some circuits, a 10K pull-up resistor should be connected from +5V to pin 4. The READ DATA STROBE output is connected to some address-decoding circuits so that input data is present on the data bus only during an external read operation.

Pin 6, the WRITE DATA STROBE, is similar in function to pin 4. The primary difference is that this pin goes low during external-memory *write* operations. Like pin 4, a 10K pull-up resistor is sometimes connected to this pin. That output is required so that external memory (or output) devices know when the information on the data bus is valid.

Pins 38 and 39 are the INS8070's two

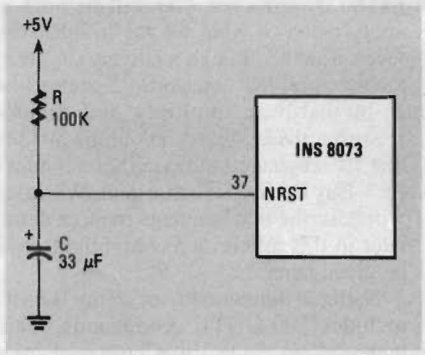

FIG. 4—POWER-ON RESET. If pin 37 of the INS8073 is connected as shown, it will automatically execute a program stored in EPROM (starting at address 8000) at power-up.

interrupt pins. Both act as interrupt-request pins. (The 8073 does not have the equivalent of a non-maskable interrupt.) In order for an interrupt to be serviced, first the input to one of those pins must go from high to low. (It is edge, not level triggered). Also, the least-significant bit

MICROINTERPRETER MANUFACTURERS

Fairchild Camera and Instrument Corp.
Semiconductor Groups
464 Ellis Street
Mountain View, CA 94042

Fujitsu America
910 Sherwood, Suite 23
Lake Bluff, IL 60044

Hitachi America, Ltd.
1800 Bering Drive
San Jose, CA 95131

National Semiconductor Corp.
2900 Semiconductor Drive
Santa Clara, CA 95051

NEC Electronics USA Inc.
Microcomputer Division
One Natick Executive Park
Natick, MA 01760

OKI Semiconductor, Inc.
1333 Lawrence Expressway, Suite 401
Santa Clara, CA 95051

Panasonic
Matsushita Electric Corp., Industrial Division
1 Panasonic Way
Secaucus, NJ 07094

Rockwell International
Microelectronic Device Division
PO Box 3669, 3310 Miraloma Avenue
Anaheim, CA 92803

Texas Instruments
Semiconductor Group
PO Box 225012, M/S 308
Dallas, TX 75265

Zilog, Inc
1315 Dell Ave
Campbell, CA 95008

in the status register must be 1. (The status register is accessible through the NSC Tiny Basic STAT function.) Pin 38 has priority over pin 39 if both interrupts occur at roughly the same time.

In addition to acting as inputs for interrupts, both pins serve as sense inputs. Here we are mainly concerned with pin 38, which will accept serial ASCII input data. The input data must first be converted to TTL levels.

The remaining pins will be briefly described in sequential order. Don't worry if the description is a bit confusing, the functions these pins provide will not be implemented in our project. Nonetheless, some of these pins must be connected to ground or +5V for proper operation of the microinterpreter. Refer to the notes given in the description of each pin for their proper use in our project.

Pin 1: ENABLE OUTPUT. The 8073 controls that output as follows. 1) If pin 3 (BUS REQUEST) is low, but the 8073 is not holding it low, then pin 1 is brought to the same logic level as pin 2, the ENABLE INPUT. 2) If pin 3 is low because the 8073 is holding it low, pin 1 goes high. 3) Pin 1 is always high if pin 3 is high. Note: Pin 1 can be left unconnected.

Pin 2: ENABLE INPUT. 1) If this pin is high, the 8073 sets pin 1 high and is denied access to the bus. 2) If pin 2 is low and the 8073 is holding pin 3 low, pin 1 goes (or stays) high and the 8073 has access to the bus. 3) If pins 2 and 3 are both low and the 8073 is not holding pin 3 low, pin 1 is set low and the 8073 is denied access to the bus. Note: Pin 2 should be connected to ground.

Pin 3: BUS REQUEST. This is the bidirectional bus request input/output. It already has been referred to. Like pins 1 and 2, pin 3 is used in direct memory access (DMA) and multiprocessing applications. This pin should be connected to a +5V power supply through a 10K pull-up resistor.

Pin 5: HOLD INPUT. The primary application of pin 5 is to allow the use of slow memories and peripherals. It is also used for single memory cycle extension. Setting that pin low causes the 8073 to extend an external read or write cycle. Note: In the burglar outwitter that we'll describe, pin 5 is connected to +5 volts through a 10K pull-up resistor.

Pins 34, 35 and 36: Flags 1, 2, and 3 outputs. These flag outputs can be set by writing into the appropriate flag bits located in the status register. These pins can be left unconnected.

Next month, we'll describe, in some detail, the language of the 8073 microinterpreter—National Semiconductor's Tiny BASIC. We will also take a look at a commercially available demo/development board, which greatly simplifies designing with this exciting microinterpreter. **R-E**