Process Control With Personal Computers

(Part 3, Conclusion)

Software needs and creating simple control systems

By Dr. H. Edward Roberts

This is the third and concluding article on Process Control with personal computers. In the first article, we laid the groundwork for the series by discussing basic control principles, computer-based process control with IBM-type PCs, and examples of computer-based controllers, focusing on DataBlocks's LINK system. Last issue discussed building the LINK interface to a standard IBM personal computer or clone, which, along with Altair-II control modules from DataBlocks, Inc., form the Modern Electronics Computer Controller. Also covered were sensors such as the thermistor, photoresistor, and resolver, as well as actuators such as the stepping motor. Finally, special interfaces were examined.

In this article, we'll explore software needs and then create a simple control system for the home and a system to automate testing of an analog-to-digital converter, designing these systems around the Modern Electronics Controller.

Software Control

The PC LINK, which adds more than 2,000 addresses to the very few available on a PC, is designed to operate with any programming language to address I/O ports. This allows a system designer to use the LINK board and external A-II control blocks in a familiar programming environment. In addition, drivers written in C and BASIC exist for many of the control blocks. These drivers, collectively called the Personal Control Language, or PCL (pronounced "pickle"), were specifically developed by DataBlocks for computer control applications. They give the user a simple way to control the application blocks with a high-level language, relieving him of the need to understand hardware/soft-

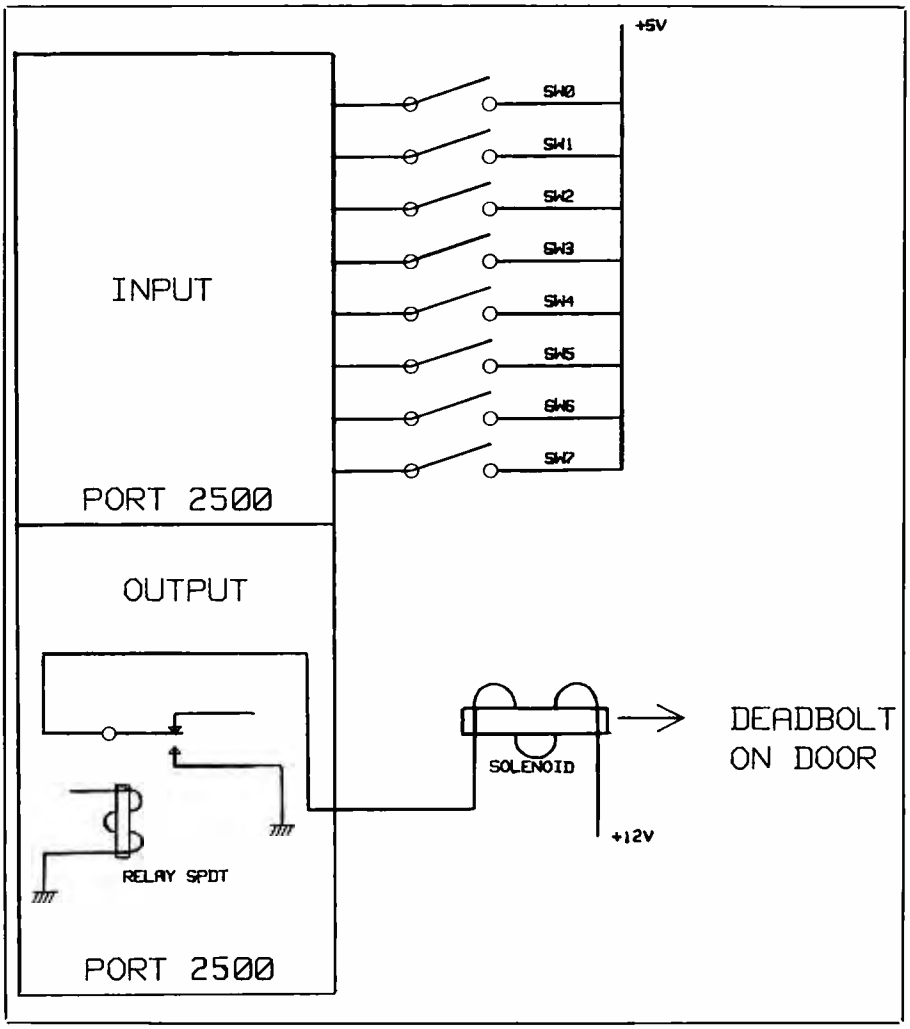

Fig. 12. Functional diagram of an electronic door lock.

ware interface details of each module. The PCL routines are contained in libraries and are accessed with simple, single line calls from the main program.

Most languages that run on a PC allow control of the input and output ports directly. For our examples, we will use standard IBM BASIC. To control a stepping motor with a dc driver block, for example, all the programmer need do is include the following line in his BASIC program:

10 STEP (port,steps,speed,nibble,stack)

where: *port* is the computer I/O port used to address the dc driver block; *steps* is the number of steps to rotate a motor, positive for clockwise motion, negative for counterclockwise; *speed* is the rate to turn the motor; *nibble* is an A or B, indicating whether the motor is connected to the upper or lower nibble of the dc driver block; and *stack* is a number between 0 and 7, denoting which of the eight possible stacks in which is contained the dc driver block.

The same line could also be included in a C program to control a stepping motor. Over 20 PCL commands exist to enable single-statement control of blocks that allow the user to control motors, read switches, monitor temperature and perform other control functions with the computer as easily as he controls his printer with the BASIC statement LPRINT.

All standard PC software is compatible with the PC LINK employed in this system series. DataBlocks also provides a real-time operating system, ARTDOS (A-II Real-Time Disk Operating System) for applications requiring real-time event processing.

To illustrate how we might control a relay block, we will write a simple BASIC program using IBM standard BASIC. The command to control an output port in BASIC is simply OUT A,D, where *A* is the address of the output port and *D* is the data to be transmitted. The following illustrates how to use this command, making the assumption that the relay block has an address of 2500. The program turns relays 0,2,4,6 on and the other relays off (relays numbered 0 through 7).

```
10 OUT 2500,&B01010101
```

That's it! The whole program is simply line 10, which is read as: Output to port 2500 the binary number 01010101. The right-most bit is 0, and it controls relay 0. The left-most bit is 7, and it corresponds to relay 7. Therefore, if bit 7 is set to a logic "0," relay 7 will be set to a logic "0" (off); similarly, a "1" will turn the relay on. For instance, the following program will turn off all relays, except relay 7:

```
10 OUT 2500,&B10000000
```

Now let's do something a little more sophisticated: make an electronic lock that will energize a solenoid-operated dead bolt on the front door of a house. For purposes of this discussion, assume that we will use toggle switches for the input. Figure 12 shows one way to wire such a circuit. A simple program to control the door lock might be:

```
10 A = INP (2500)
20 IF A = &B11000101 THEN GO
TO 50
30 OUT 2500,&B00000000
40 GOTO 10
50 OUT 2500,&B00000001
60 GOTO 10
```

In this program, we assume that both our input and output ports are at the same address, that is, 2500. Also note that the combination to the lock is 11000101. This means that switches 0,2,6,7 on the others are off. The program operates as follows:

1. First, the computer will input the setting of the switches in line 10; that is, read the switches.
2. The number read in line 10 is compared with the combination number, 11000101.
3. If the numbers match, the program will jump to line 50 and turn on relay 0. If the numbers don't match, the program turns off all relays (line 30).
4. The program will continue to cycle until stopped due to the GOTO 10 statements.

A typical PC might be able to perform the above operations several thousand times a second. This means that this routine could be "buried" in some larger program and checked occasionally, say, every 200 milliseconds. Since each check would take less than 1 millisecond, the computer will still have 99.5 percent of the second left over to do other jobs. These times are approximate, but they will give you a feel for the capability of a computer.

The lock example given above is interesting from a number of standpoints. It shows how multiple unrelated inputs (switch closures) affect an outcome. Also, by changing the number required for a match, you could make the system "adapt" to various situations.

Though number entries have been shown as binary bits, such as 11000101, the computer is just as happy to compare switch settings with decimal numbers. We could have entered 197 instead and the program would have worked exactly the same. The reason for this is that the computer ultimately compares or operates on all numbers as binary numbers. I find that in some applications it is easier to think of switch settings if I enter them in binary, but in many other applications using decimal numbers or sometimes even hexadecimal is useful. While it is not a requirement to learn binary numbers to work with control computers, anyone who wants to be really literate in computer electronics should have some familiarity with binary.

Designing Systems

Using a personal computer and controller equipment makes it easy to design and install systems to provide comprehensive control of home or office security, environment, and

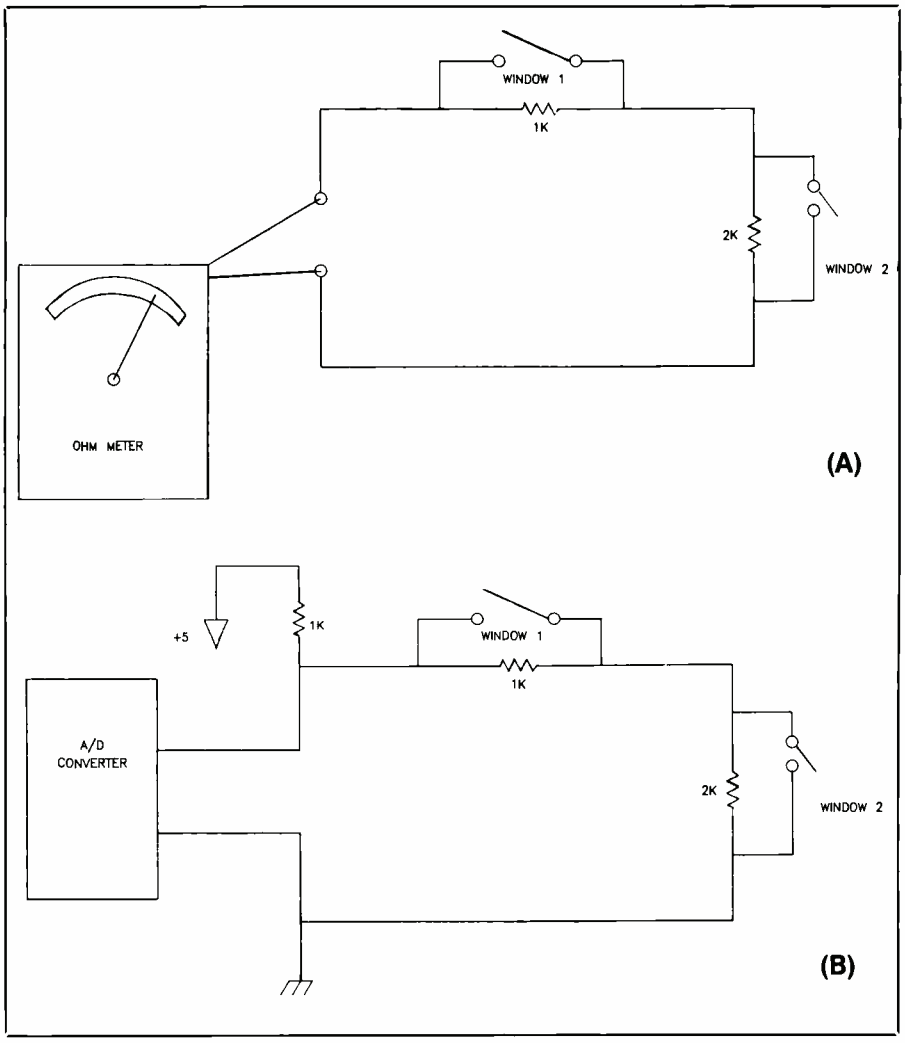

Fig. 13. Detectors for multiple windows (A) and an automated open-window detector (B).

disaster prevention. Let's discuss some approaches to doing this.

Monitoring doors and windows to detect when one is open, for example, can be as simple as wiring a resistor across a switch at each door or window to be monitored by a computer. An open-window detector's resistance can be monitored with an ohmmeter. Closing the window causes a contact switch to short out the resistor, resulting in a zero reading on the meter. Opening the window removes the short and the meter then reads the resistor's value. Let's expand this concept by wiring two windows in series, as shown in Fig. 13A. Now the meter will not only detect when one of the windows is open, it can also reveal the window's location if the value of the two resistors is different. Accordingly, if window 1 is open, the reading will be 1k, and if window 2 is open, the resistance is 2k. Of course, a 3k reading will indicate both windows open.

Extending this one step further, we can automate the system by replacing the meter with an analog-to-digital converter circuit, as shown in Fig. 13B. Here, the A/D converter reads 0V when both windows are closed since all the resistors, except *R1*, are shorted. If window 1 is open, the A/D reads 2.5 volts, while a 3.3-volt reading is detected if window 2 is open. If both windows are open, no current will flow in the circuit, and 5 volts will be measured by the converter. It is easy to see that this approach could be extended to include all windows and doors in a home or office.

The principles of environmental control are also easy. You can conserve energy simply by reducing the temperature in the house during the time of day when no one is home. This can be accomplished by placing a relay in series with the furnace thermostat control wire, as depicted in Fig. 14. A computer can be programmed to open the relay during the time of day when no one is home to allow the temperature to decrease.

It would be advisable, especially in very cold weather, to add some feedback to the system by monitoring the temperature with a thermistor circuit and an A/D converter. If the temperature decreased below a desired level, the computer would reactivate the relay and run the furnace until the temperature was again at an acceptable level. This system is also useful in the summer to reduce electricity used by an air conditioner. In addition, this simple conservation approach can be used to reduce energy wasted by an electric water heater.

Recreation and relaxation offer many opportunities for productive automation. A good example is using energy from the sun to heat the water in swimming pools and hot tubs. To have an effective solar heating system, a process control system must monitor several parameters and control several actions. For instance, the system must be able to determine if the solar isolation is high enough to increase the water temperature rather than cool it. This can be done by programming the controller to compare the water's temperature as it leaves the heater with the water's temperature in the hot tub. If the water leaving the heater is cooler than upon entering, it's time to abandon the solar heater until a sunnier day.

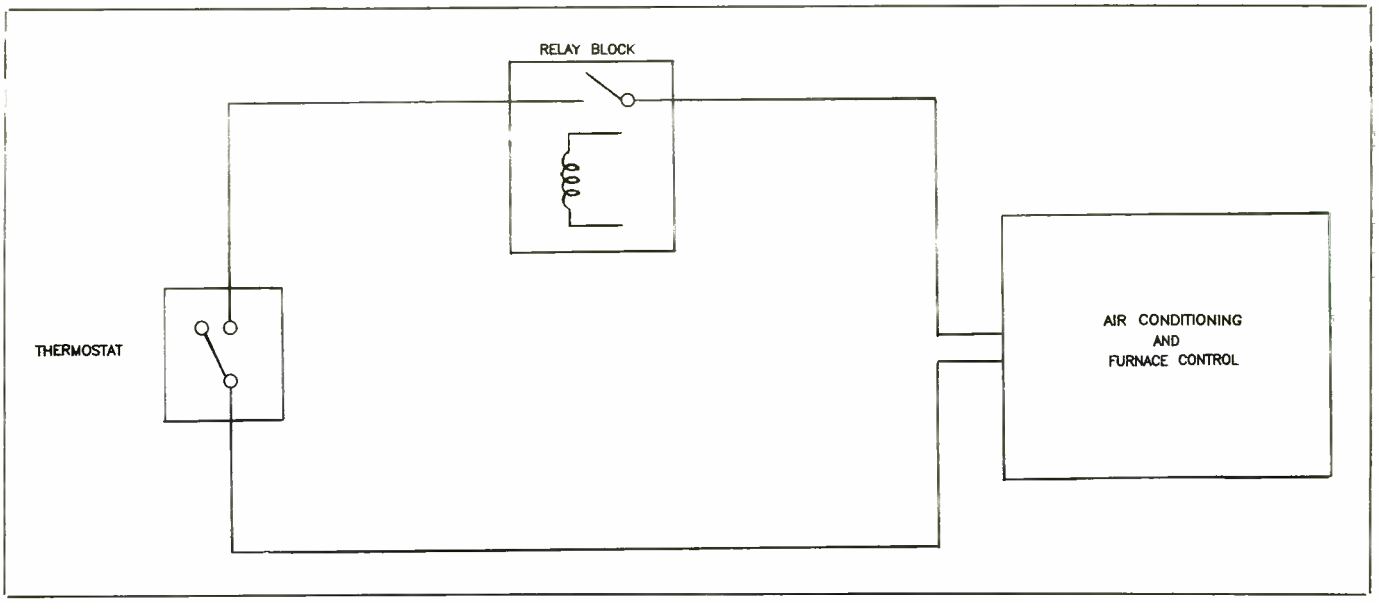

Fig. 14. Schematic details of a thermostat control system.

The water in a pool or hot tub should also be monitored to ensure that it does not become too hot. If it does, the computer must activate valves to stop the flow through the heater or activate a motor that will reorient the solar collector to reduce the amount of energy transferred from the sun to the water. The controller should also open and close valves to replenish the water lost due to evaporation and spillage. In addition, it should run the filtering systems periodically when the pool or hot tub is not in use and continuously when it is being used. These functions are easily implemented using the Controller previously described.

Monitoring indicators of impending disaster is also a very useful function that is easily performed by our controller system. For example, detecting when a smoke alarm has been triggered can be accomplished by monitoring the alarm's buzzer with the circuit shown in Fig. 15. When the alarm is activated, the voltage across *C1* increases. This increase is measured by an A/D converter, which causes the controller to dial the fire department through a computer modem. The modem circuitry informs the controller when the fire department phone has been answered, and the controller can activate a speech-synthesis block to give the fire department dispatcher information on the fire's location.

To illustrate the rudiments of designing with the controller, let's now design a system that brings together some of the home-automation ideas discussed. The first step is to define the system requirements. Let's put together a simple system, illustrated in Fig. 16, which does the following:

1. *Provides security.* (a) Monitor windows. (b) Turn on a floodlight if a window is opened (requires easy override so window can be purposely left open. (c) Make house look lived-in when occupants are gone (requires easy means to activate/deactivate).

2. *Reduces heating and cooling load during the day.* (a) Should defeat the setback on Saturday and Sunday (or any days desired). (b) Requires feedback to prevent the house from becoming too hot or cold during the setback period.

3. *Controls a solar water heating system for a solar hot tub.*

The control system we'll use consists of an IBM PC or clone, a DataBlocks LINK and a stack of the control blocks necessary to meet the control requirements. One block we need on the stack is a DataBlocks KAD-710, a 16-channel A/D converter, to monitor some open-window detector circuits such as those cited. We'll also add a Powerhouse X-10 universal interface to the controller stack to turn on a floodlight if the computer detects a window being opened. The Powerhouse X-10 system will also be used to control lights

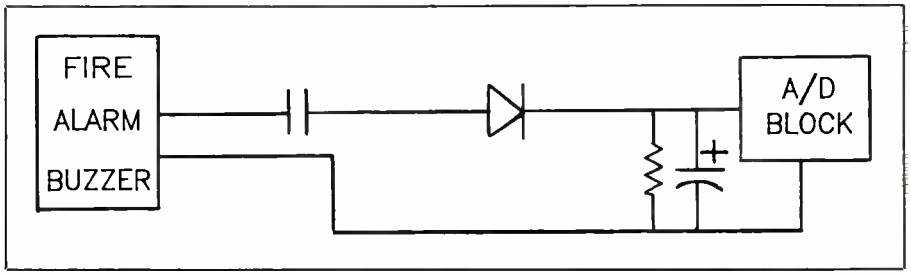

Fig. 15. A fire alarm monitor arrangement.

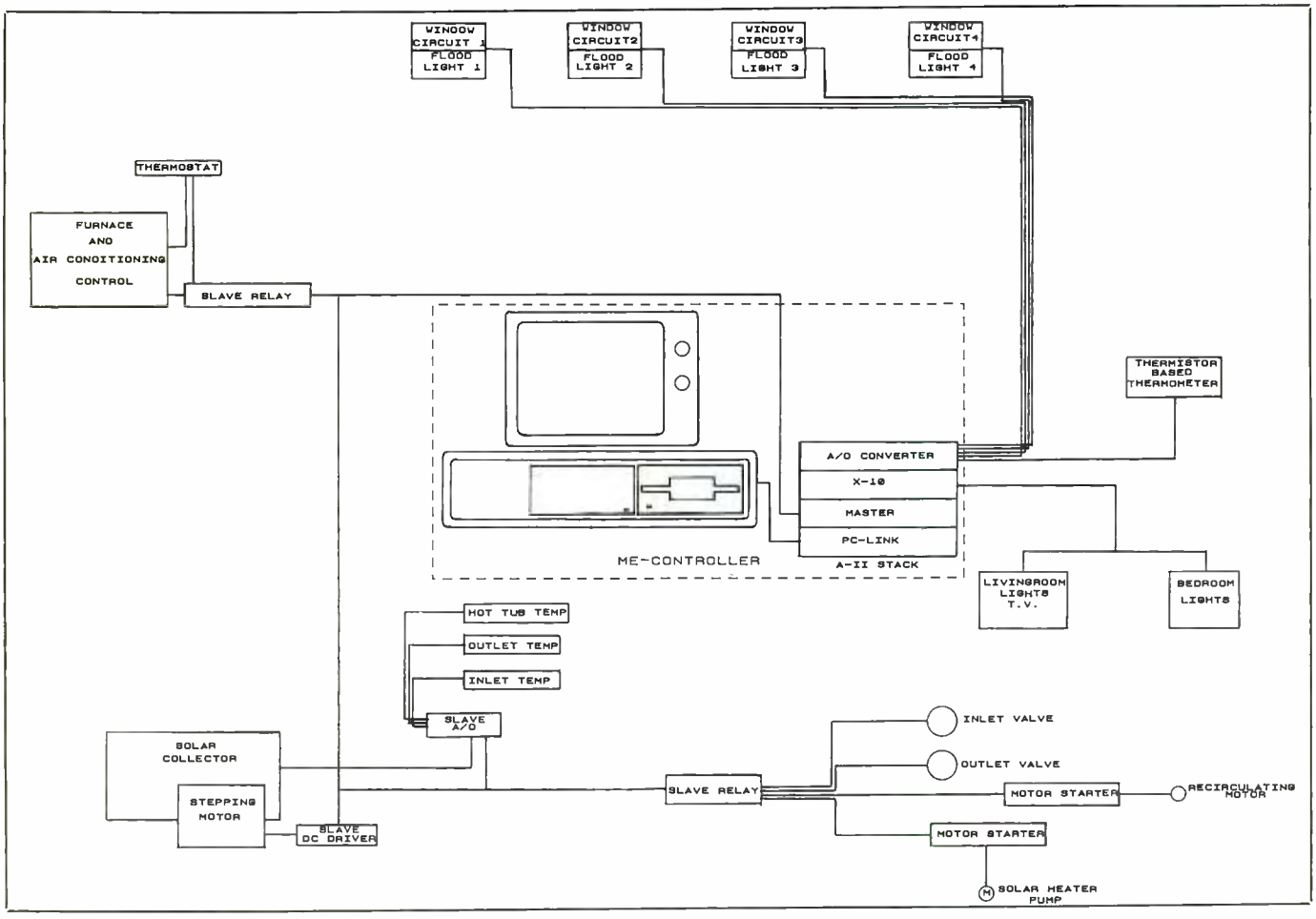

Fig. 16. A typical home automation system.

in various parts of the house to make it look lived-in when unoccupied.

A DataBlocks Master Control Block will also be included in the stack. This block will be used to send and receive data between several slave blocks needed to complete our control system. One of these is a slave relay to interrupt the thermostat and shut off the air conditioner or heater when no one is home. To prevent the house from getting excessively hot or cold when it is unoccupied, we should add some feedback to this part of the system. We'll do this by using a channel of the A/D converter already on the stack to monitor a thermistor circuit such as the one shown in Fig. 17.

You'll recall from Part 2 that a thermistor exhibits a resistance that varies with temperature. Therefore, as the temperature in the house changes, the thermistor's resistance also changes and causes the voltage across *R1* to change. When voltage across the resistor moves outside a tolerance window that represents the maximum and minimum allowable temperature, the computer sends a message through the master to the slave relay, causing the relay to close. This will allow an air conditioner or furnace to run until the temperature changes enough to cause the voltage in the thermistor circuit to again indicate that the temperature is within acceptable limits.

The solar water heating for the hot tub will also require slave blocks. Let's assume we will install a parabolic solar collector to transform the energy from the sun into heat. This type collector should always be pointed toward the sun and, therefore, we need a control circuit to rotate the collector as the sun's orientation to the collector changes during the day. As observed in last month's article, a stepping motor provides excellent rotational control without requiring explicit feedback. This makes the stepper a good choice to drive the solar collector. The stepper is driven with a dc driver block; since the motor will be located several feet away from the Controller, a slave dc driver block is the best choice.

One of the most important things to consider when using a solar water heater for a hot tub is water temperature. As in the case of monitoring temperature in a house, the thermis-

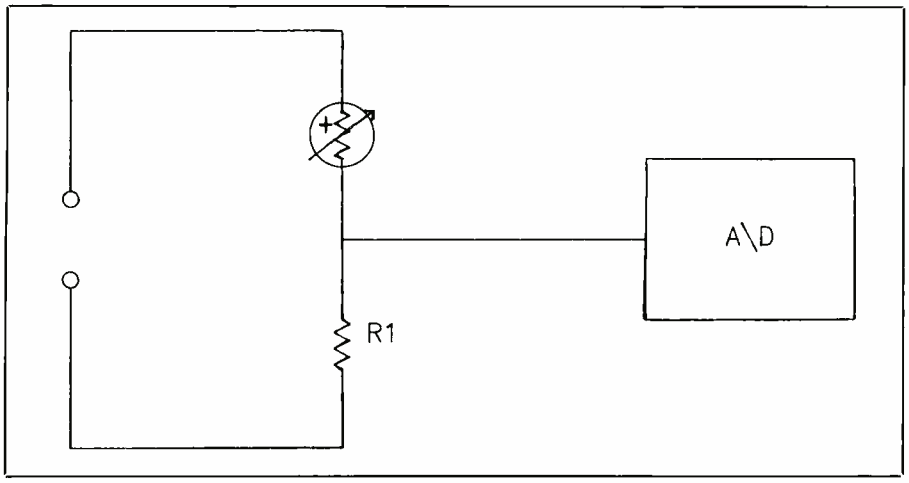

Fig. 17. A thermistor-based thermometer.

tor circuit shown in Fig. 17, monitored with an A/D converter, should work well. Since the hot tub is likely to be remotely located from the controller, we will again select a slave A/D converter. In this case, we'll use three channels of the A/D converter to monitor inlet temperature, outlet temperature and the temperature of the water in the tub.

It would also be a good idea to use a channel of the A/D converter to monitor a switch on the solar collector hardware. This switch should be mounted so that it is closed when the collector is facing about 20 degrees below the east horizon, providing a starting point from which to orient the collector. This will enable the software to re-establish the correct collector orientation if the system loses power. This is the "parked" position of the collector.

Several valves must be opened and closed, too. Also, the hot tub recirculating motor and solar collector water pump must be controlled to complete our hot tub automation project. This will best be handled with a slave relay. Current required for valves is low and, therefore, they can be controlled directly with the relay. However, depending upon the size of the recirculating motor, the current requirements, especially the starting current, may exceed the 6-ampere capacity of the relays on the relay block. It would be better, therefore, if we used a motor starter, which has lower current requirements, to switch this motor. The pump for the solar collector may not require more current than a relay can switch. In this case, it can be controlled directly by the relay. However, depending on the size of the collector, the size of the pipe to the collector, and the height to which the water must be pumped, the pump may indeed draw enough current to also require use of a motor starter.

Now that the hardware portion of the design process is done, we'll design the software part of the system. There are probably several approaches to designing software for this automation project. The easiest

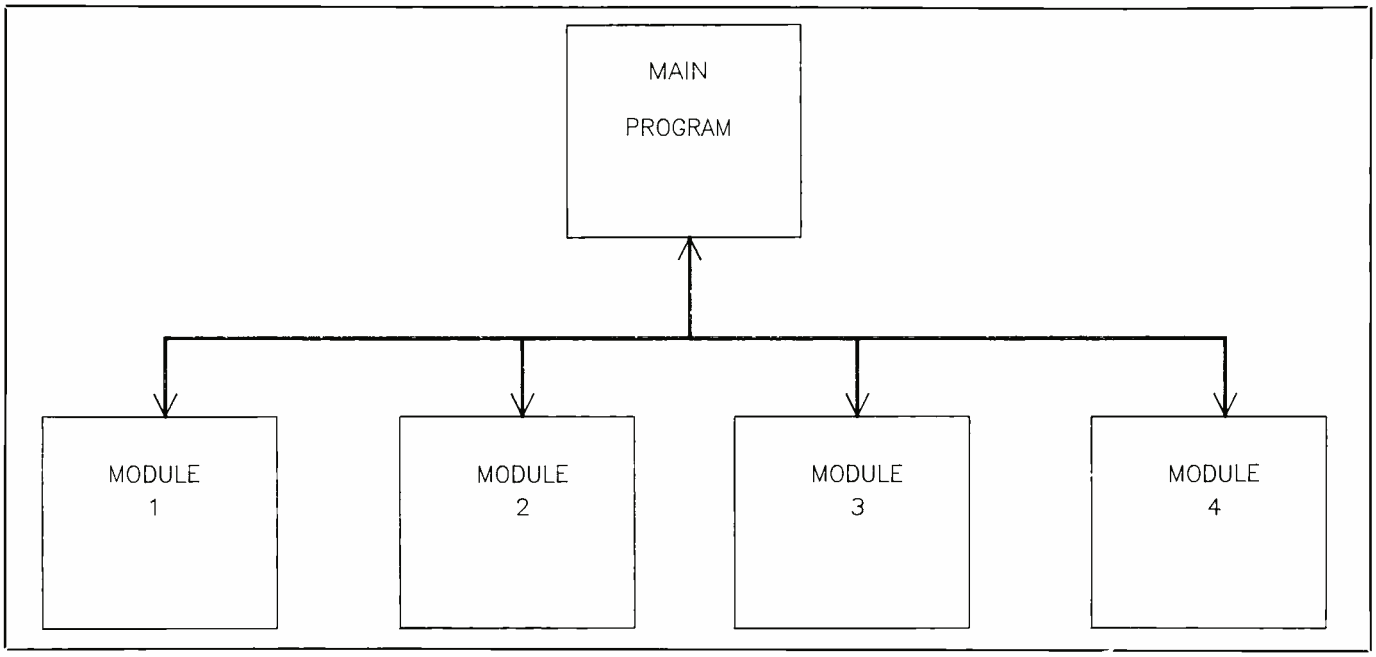

Fig. 18. Typical Software Functional Flow diagram.

to contemplate for our example is one in which the software sequentially branches to modules that perform one of the functions previously described in the system's requirements. Figure 18 depicts this as a main flow diagram and four modules.

Such a modular approach is useful for several reasons. First, it allows very complicated software to be decomposed into simple components to facilitate development. Although our example is not particularly complicated, the modular approach makes it even easier. Second, a modular approach allows additional modules to be added without significant software rewrite. Finally, it makes the software much easier to test and debug.

Once in a module, the software compares the current time with preset start and stop times for the various functions. The preset times, along with other fixed parameters, are called the data base and should be stored in a file on disk and read into the program as necessary. This makes it easy to change the parameters as requirements change. It also permits access to several sets of parameters to accommodate different circumstances, such as seasons of the year and vacation schedules. The files could easily be made and stored with a utility program written for that purpose.

Figure 19 illustrates the software control flow. The initialization routines are executed first. They establish the operating environment for the program by reading a parameter file from disk and storing the data in data-base memory so that it can be used by the rest of the program.

You may choose to store the data in separate variables, or you can group data functionally and store it in arrays. Included in the parameter file are such things as start and stop times for each function, maximum and minimum temperatures, and X-10 addresses and control codes. Depending upon your approach and

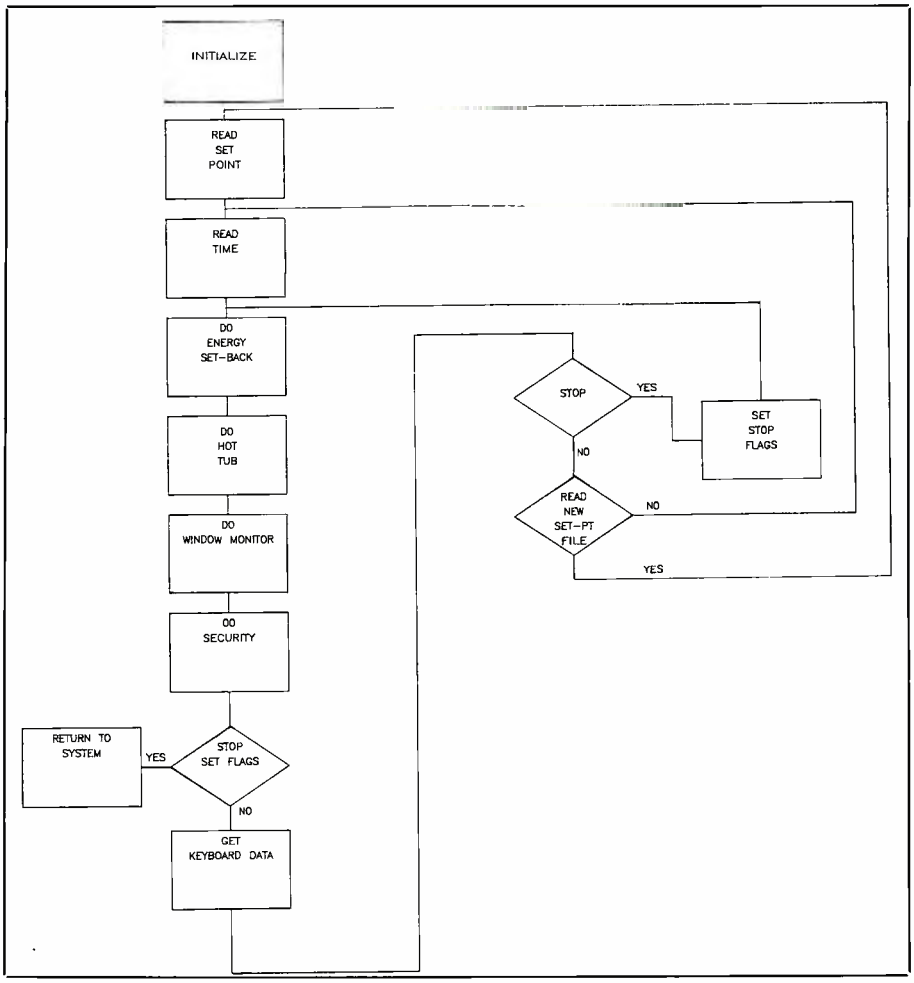

Fig. 19. The Main Program Flow diagram for the system described in the text.

the system enhancements you wish to include, you can add to or subtract from this list when you develop your software. In addition to data-base initialization, the initialization routines may contain other software pertinent to getting the system ready to control your home or office. For example, you might wish to devise a test to ensure that each of the hardware blocks is working when the system is powered on. The software to perform these tests will be included in the initialization routines.

After initialization, the main program sequentially transfers to each of the service modules. The first service module is the one that controls the energy setback relay on the furnace and air conditioner relay. Figure 20 shows the thermostat's logical program flow through this module. Upon entry into this module, the software checks a flag to determine if the routine has been activated. That is, whether the start-time criteria for this activity, defined in the data base, has been met on a previous pass through the module. If not, it tests the current time against the start criteria to see if it is time to activate the energy setback function. Until the start time is reached, the software simply returns to the main routine to continue servicing the other modules.

When the start time is reached, the active flag is set and the software executes the rest of the code in the module. The first step is to see if it is time to stop. Of course, on the initial time through the routine, it will not be time to set the stop; but we do need to

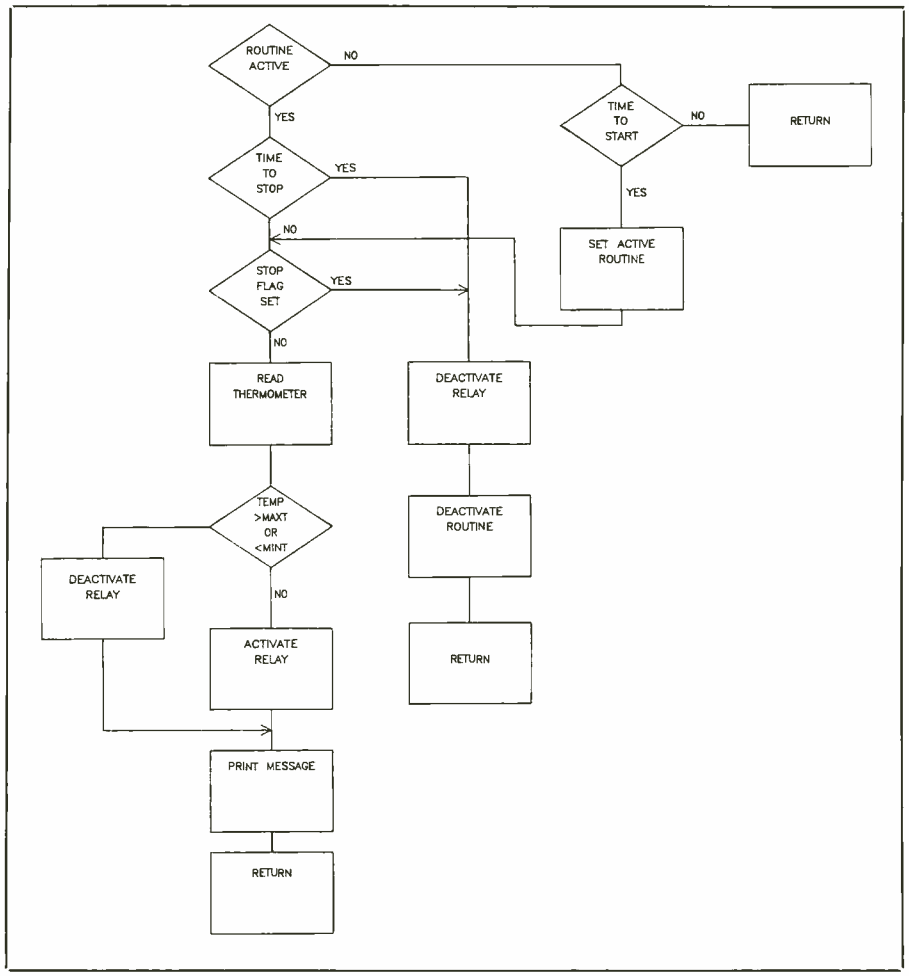

Fig. 20. A Thermostat Set Back Flow diagram.

make this check later. Moreover, the programming logic is simpler if we check each time through.

The next step in the flow diagram is to check if a stop flag has been set. This flag is the one that is set from the keyboard to interrupt the normal flow of events. For example, if you come home from work early and the energy setback routine is in operation, you need an easy way to deactivate the module and make the house more comfortable. Setting the stop flag from the keyboard will initiate an orderly shutdown and allow you to do this. If the stop flag is set or it is time to stop, the software deactivates the relay controlling the setback, deactivates the screen and returns to the main program.

If neither of the two conditions that cause the software to exit the module are true, the program reads the thermometer. The temperature is compared with two data-base values, MAXT, the maximum temperature you are willing to let the house attain, and MINT, the minimum temperature you want.

If the temperature in the house is determined to be greater than MAXT or less than MINT, the program deactivates the setback relay, allowing an air conditioner or furnace to run until the temperature is again between maximum and minimum. If the thermometer reading indicates that the temperature is between extremes, the software activates the setback relay to to conserve energy and then returns to the main routine. At this point, you might question turning on a relay that may already be on. Since there is no mechanical reaction, it makes no difference. You could test to see if the relay had previously been turned on, but this complicates the software unnecessarily.

The second module our software will service is the hot tub. As illustrated in Fig. 21, the software in this module reads the tub's temperature as soon as the start time has been reached. The temperature is compared with MAX, a data-base value that you set to establish the hottest temperature you want the hot tub to reach. If the temperature is equal to or greater than MAX, the recirculating pump is turned off, the solar collector is pointed away from the sun, the water pump is turned off, and control returns to the main program. If the temperature in the tub is less than MAX, the recirculating motor and water pump are started, and the collector is pointed at the sun.

A word about controlling the collector is in order here. Temperatures in a parabolic solar collector on a sunny day can become very hot. Therefore, in our application it is important that the collector be pointed away from the sun when water is not circulating through it. This will prevent water that is extremely hot from entering the hot tub when the water pump is first started. Recall that we made provisions for a park position for the solar collector by monitoring a switch closure with the slave A/D converter. Turning the collector to the parked position when the water pump is off will prevent the water from becoming overheated.

Let's also examine how we'll keep the controller pointed toward the sun when we have the water pump on. We could install a sensor that signals the computer when the collector is pointing toward the sun. Remember, however, that the rotational position of the stepping motor can be determined by keeping track of the number of steps sent to it. We also know

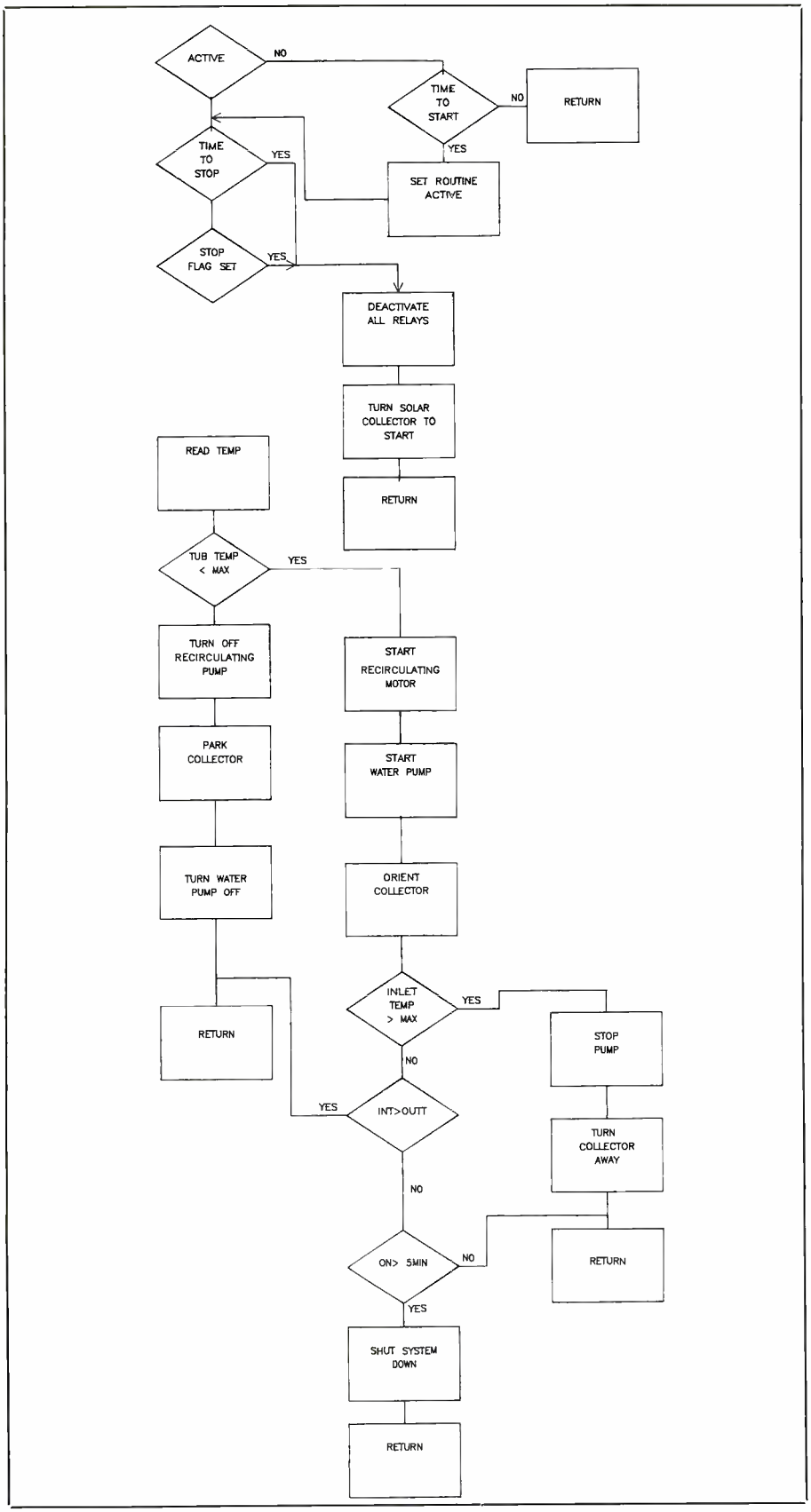

Fig. 21. A Hot Tub Software diagram.

that the location of the sun is very predictable if we know the day of the year, time of day and our location. It simplifies our design, therefore, if we compute how much collector rotation is needed to point toward the sun and the number of steps we need to send to the stepping motor to turn the collector that amount.

Once the collector is oriented, check the inlet temperature to make sure the incoming water will not burn anyone in the tub. If the water is equal to or greater than the data-base value MAX, we immediately stop the water pump, turn the collector away from the sun and return to the main program. If the inlet temperature is less than MAX, we check to make sure it is greater than the outlet temperature. If it is, the system is heating as it should and we return to the main program. If it isn't the software checks to make sure the system has been on long enough (5 minutes) to get the cold water out of the pipes and allow the collector to become heated. If enough time has passed, the software starts a wait timer for, say, 10 minutes to prevent the turn-on sequence from being executed every few seconds. Then the system is shut down to wait for a sunnier time. Otherwise, we return to the main routine to service the next module and give the heating system a chance to warm up.

The next module scheduled for the main routine is the window monitor routine. The software flow for this module is shown in Fig. 22, the Window Monitor Software Flow Diagram. Once it is time to activate this routine, each of the four channels of the A/D converter in the controller is read. If the voltage indicates any open windows, a flag is set to identify the circuit and window that is open. The software then checks to see if a security flag is set. This flag indicates to the software whether you wish to activate the flood lights and sound the alarm when the window is open or whether you only

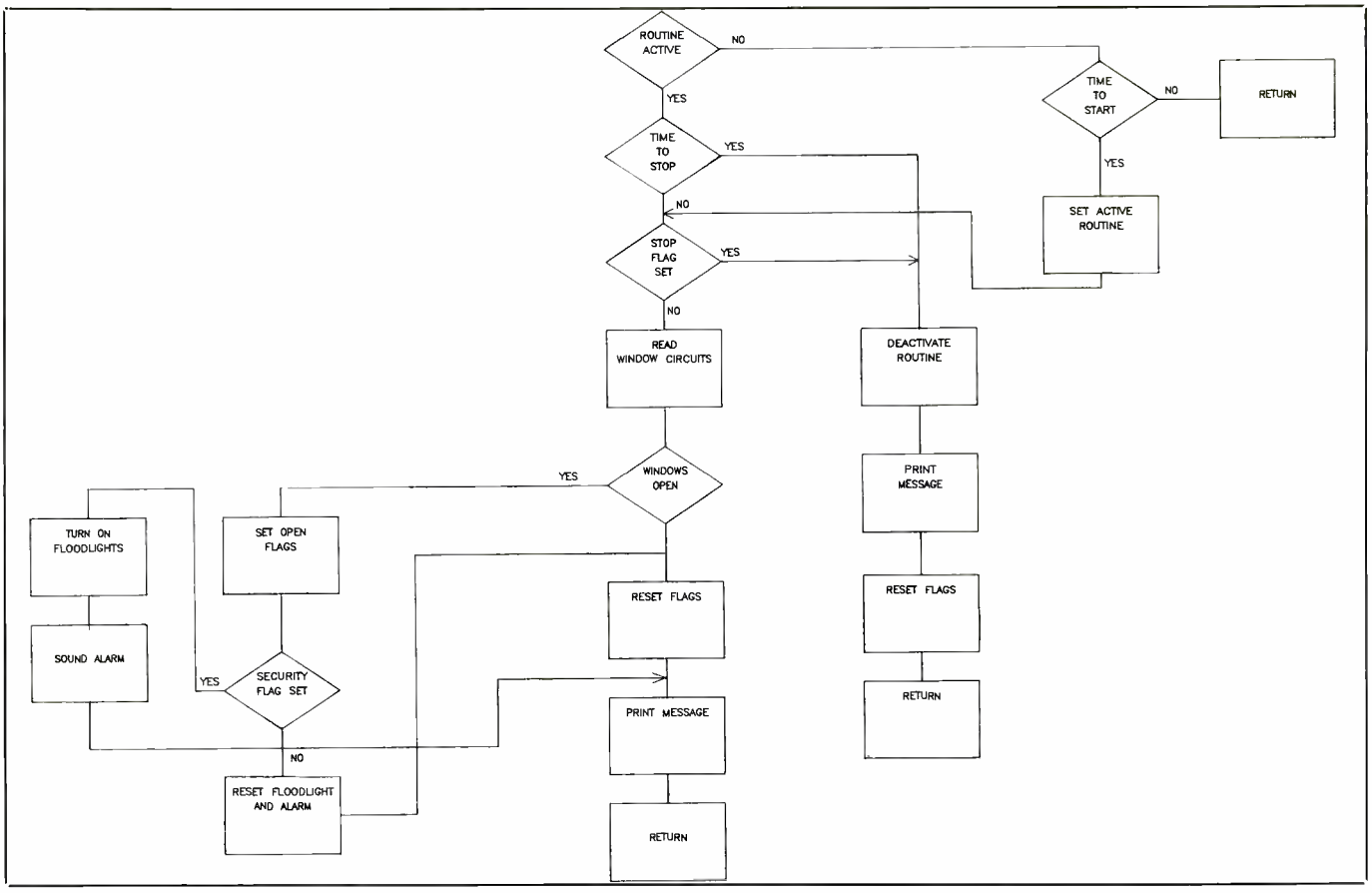

Fig. 22. The Window Monitor Software Flow diagram.

want to display a message on a video monitor. You will probably want to activate this flag based upon time of day, with the capability to change it from the keyboard. This will give your system the flexibility to provide protection during periods of non-scheduled absence from a house.

If the flag is set, the floodlights and alarm are turned on, a message that the window is open is displayed, and the software returns to the main routine. If the flag is not set, the software turns off the floodlights and alarm, displays an open window message and returns. The alarm for this circuit can be selected from a variety of devices. You might choose to turn on a radio or a stereo system, flash lights off and on, or install a loud buzzer.

The last software module serviced by our home control system is the one that makes the home looked lived-in when it is unoccupied. The key to successful implementation of this portion of the system is how natural the pattern of lights, TV, stereo, etc., is to an external observer. For example, if a bed room light flashes on and off too rapidly or always comes on at exactly 2:37 a.m., it will be obvious to an observer that the lights are not operating according to normal living patterns. Establishing appropriate timing patterns, then, will require some planning and testing to accomplish successfully. Making the start and stop times available to the software is probably best done by forming an array containing a table of start times, stop times and the appropriate address of the device or devices to control.

Referring to Fig. 23, the Occupant Activity Software Flow Diagram, each time software enters this routine it checks to see if it's active. If it isn't, it checks to see if it is time to activate it. If not, the software returns to the main program. Otherwise, the software searches the table of start and stop times to determine which devices need to be turned on and which need to be turned off. It then outputs the appropriate X-10 commands for each device in the table and returns to the main program.

The main program has now serviced all the modules instantly, as far as humans are concerned, and is ready to begin over again. Before it does, however, it determines whether the stop flag was set on the previous pass, If it was, this control program stops executing and control passes to the Controller's operating system. If the stop flag has not been set, the program determines if there

has been keyboard activity during the previous pass.

Our software discussion has been generic in that we have not made any assumptions that favor or preclude any programming language. One of the attractive features of the Controller being used here is that it is based upon familiar computer hardware running familiar software. As a result, you can use the programming language that is most comfortable to you. The only thing you might want to consider when you choose your programming language is the ease with which the language supports manipulation of the I/O ports. Obvious good candidates are BASIC, C and Pascal.

Testing Environment

So far, we've confined our discussion to control systems around the home or office. The Control system's versatility, however, makes it equally capable of control work in a laboratory or test environment, as follows.

Assume we need to perform a functional check of A/D converter blocks as they come off a production line. The requirements of the test are:

1. Test the conversion accuracy over a 0- to +5-volt range.
2. Verify operation of all 16 input channels.
3. Test operation and accuracy of the programmable gain amplifier.
4. Test that conversion time is within specification.

Further, we need to store the data acquired during the test and produce a hard-copy report when the test is concluded. Figure 24, an A/D Test Configuration, shows the hardware used to perform the tests. In addition to the PC and the LINK, the Controller is configured with a line-printer block, an IEEE-488 block, timer/counter block and a dual D/A converter. A digital multimeter with an IEEE-488 interface, a line printer and a magnetic tape recorder are also part of the test configuration. The

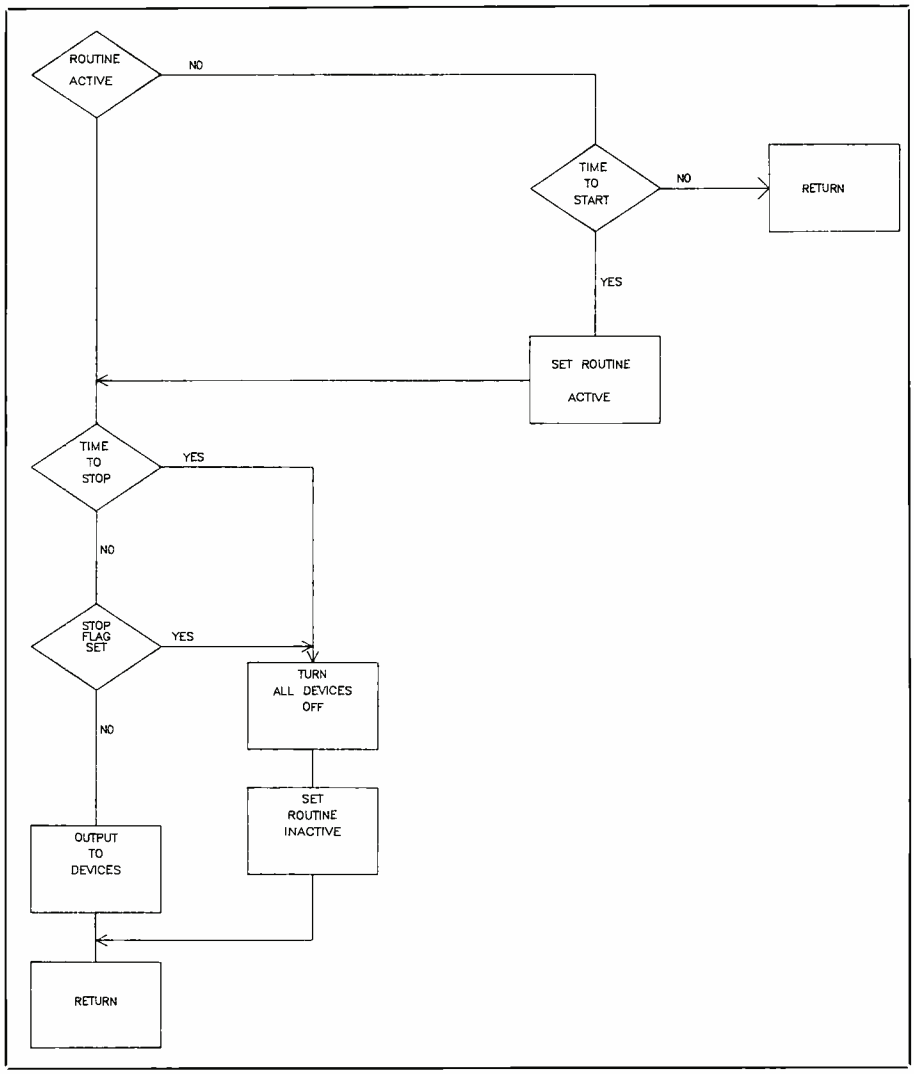

Fig. 23. The Occupant Activity Software Flow diagram.

A/D block under test is included on the Controller stack.

The dual D/A block is used to produce a programmable voltage that is fed to the A/D input channels. This voltage will be changed by the Controller during the test to determine if the A/D converter under examination can digitize across the range of 0 to +5 volts. This range is important because it generally repeats minimum and maximum signal strengths sent to and from I/O ports.

Input signals to a computer are as-

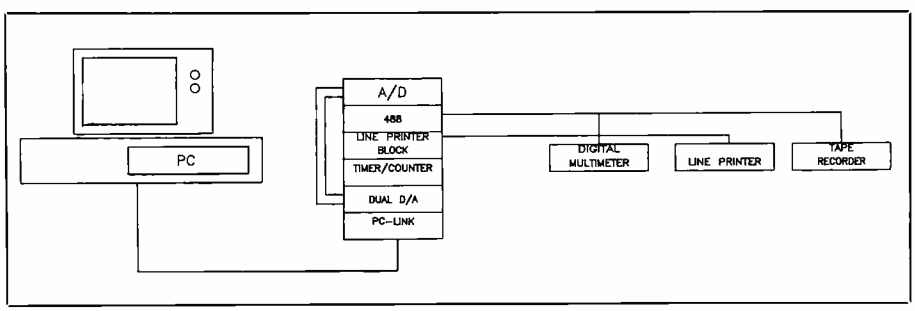

Fig. 24. An A/D converter test configuration.

sumed to be a "1" if they are greater than approximately 3 volts and assumed to be a "0" if they are less than 1 volt. If you remember this rule, it is relatively easy to interface to a computer from a signal standpoint. But there is one more detail since all the devices in the system use the same eight lines to transmit and receive data. We simply cannot connect our input signal directly to the data bus. If we did, we would lock up the bus. What must happen is that the input controller places a switch between the incoming signals and the bus. This switch is kept open until the correct conditions are met; that is, right address or IOR. Simple!

If input signals to the computer are greater than the allowed amount, you may have to use a voltage divider to lower them to acceptable levels. Also, if there is a chance of a high-voltage spike occurring on the line, an opto-isolator is probably a good device to use. In some cases where the polarity is wrong or the signal is noisy, more complex signal conditioning may be required. Another good point to remember is that input signals should be off prior to turning on the computer. Unfortunately, it is not always possible to ensure that the inputs will be off prior to power-up. In those cases, serial resistors and clamping diodes should be used.

The IEEE-488 block controls the digital multimeter functions and acquires data from the multimeter. This data serves as a control on the test results by providing a redundant measurement of the input to the A/D converters. In addition, this block controls the tape recorder, providing a permanent archive of test results.

The timer counter block is used to determine the conversion time of the test A/D converter. One of the counters in this block is present to start a count when the A/D is given a convert command. The accumulated count is then read at the end of the conversion process to determine conversion time.

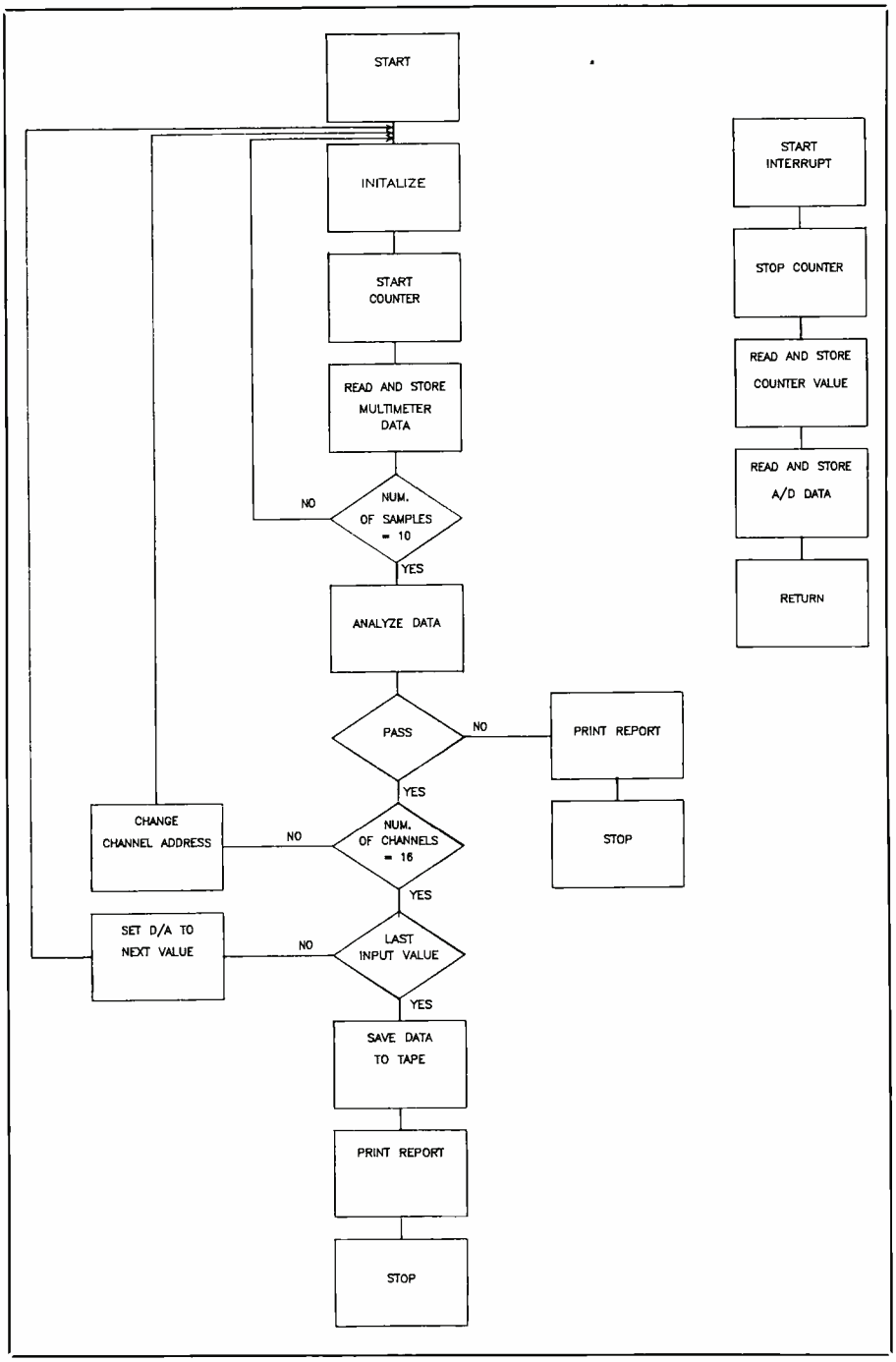

Fig. 25. The Test Set Software Flow diagram.

Software for this control example has to be designed somewhat differently from that of the house-control example by taking advantage of the Controller's interrupt capability. This means that instead of continuously sequencing through software to determine if certain events have happened, we will let the device inform us when they are happening by signaling the controller through the interrupt system.

The flow diagram for the test software is shown in Fig. 25. The software begins by performing the necessary system initialization. This will

include running any desired self-test routines, sending initialization sequences to the IEEE-488 and line-printer blocks, setting up the interrupt system and establishing a data base of constants to be used by other parts of the software as needed. Once initialization is completed, the software proceeds to the A/D converter test.

The D/A output is set to the first test value in the data base. The timer in the clock circuit is started and the A/D is commanded to digitize the voltage on the first input channel. When it has completed the conversion, the A/D block signals the computer through the interrupt system. This causes the Controller to stop the counter, input the data from the A/D, and store it in memory. When it has finished, the Controller sends a command to the digital multimeter, causing it to measure the A/D input voltage. The voltmeter measurement and the time from the counter are placed in the Controller's memory. This sequence is repeated 10 times for the first input channel. At the end of the tenth sample, the analog input channel address is incremented to the second channel: the 10 measurements from the A/D converter, counter and digital multimeter are then repeated. This process continues until all 16 channels are tested at the current input voltage.

The Controller now sets a new value into the D/A converter and the 16 input channels are commanded to measure the new input voltages 10 times each. This process continues until all the data-base values between 0 and +5 volts have been measured by all 16 channels on the A/D 20 times. This completes requirements 1, 2 and 4 of our test program.

The last requirement to be tested is the programmable input amplifier. There are three values of gain for this amplifier: 1, 10 and 100. Since the gain was set at 1 for the previous set of tests, this part of the gain test is already complete. The software now programs the amplifier for a gain of 10 and sets the voltage from the D/A converter to 0.4 volt. A sequence of 10 measurements from the A/D converter and from the DMM are taken and stored in memory. After the measurement sequence is completed, the data in each sequence is averaged, and the average of the A/D converter is divided by the average from the meter. The result should be 10. If it is (within some tolerably accepted error), the A/D converter gain is programmed to 100 and the input from the D/A converter is set to 0.04 volt. The average of 10 samples from the D/A converter is divided by the average of 10 samples from the meter, and the result is compared with 100. If it is 100 (again with an acceptable error), the A/D converter passes the amplifier gain test.

During each sequence in the test, data acquired from the A/D converter, the digital multimeter and the counter are analyzed as desired to ensure that the A/D converter is functional. At the end of each sequence, the analyzed data is transmitted over the IEEE-488 interface and stored on disk or tape. In addition, a short pass/fail report for the unit is printed on the line printer, concluding the A/D test.

Conclusion

This series illustrated how computers can be used for a variety of control applications. Whether you want to generate a control signal based on input signals or collect real-time data for process control, quality control, research, etc., computer technology makes it easier and speedier.

Using an IBM PC or compatible computer, DataBlocks' LINK system of interface and application-specific, encased modules, a host of tasks can be simplified, as we've shown. This methodology invites you to push a PC's performance and utility far beyond its typical uses at moderate cost. For more LINK system info, call toll-free: 800-652-1336.