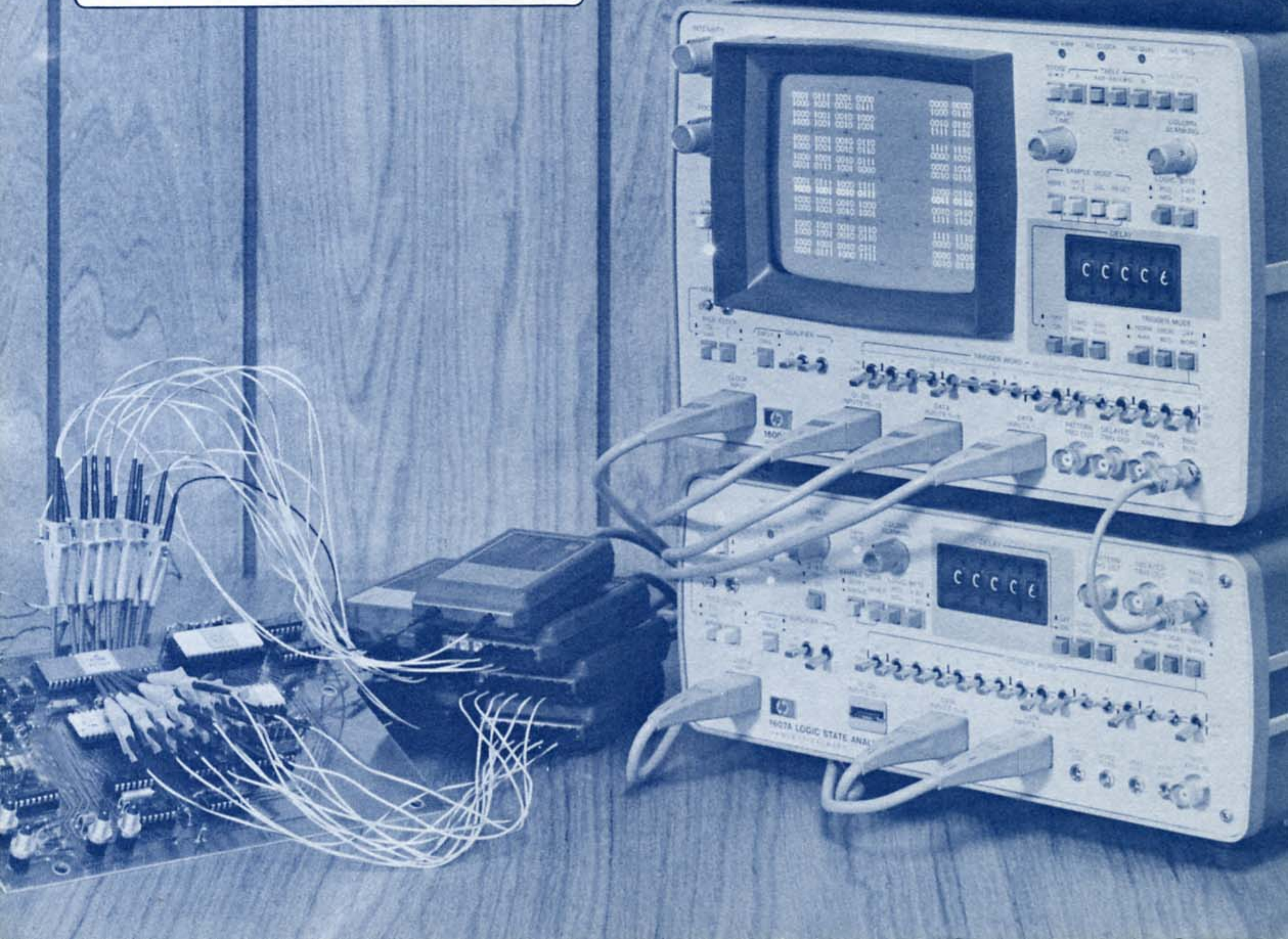
APPLICATION NOTE 167-19
DATA DOMAIN MEASUREMENT SERIES

Systematic "turn-on" of microprocessor systems using logic state analyzers.

reprinted from Electronic Design Magazine



Bring up your μP "bit-by-bit." With a systematic, section-by-section approach you can test both hardware and software in less time.

Reprinted from Electronic Design 15, July 19, 1976, copyright 1976 by Hayden Publishing Company, Inc.

Unless you enjoy spending weeks or months in unraveling complex puzzles, take a systematic approach when you develop μP-based systems.

Two basic procedures underpin the approach: turn the system on one piece at a time, and debug software and hardware together. The tool that allows all this is the logic-state analyzer.

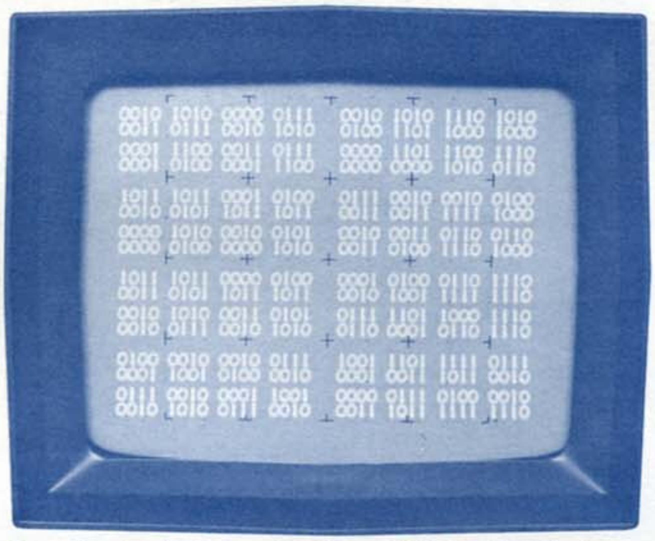
With the analyzer, you can develop a μP system with the same technique you usually use to develop a cascaded amplifier. That is, you usually turn on and check out the amplifier one section at a time by injecting a signal into the input and measuring the output of each section with a voltmeter, spectrum analyzer or oscilloscope. In turning on μP systems, analyzers offer a similar approach, and so ease the development process.

In operation, analyzers measure information transfer—program addresses, program instructions or any data that appear on any of the busses or ports within the μP system.

One analyzer can display sixteen 16-bit data words at one time, or sixteen 32-bit words with a companion analyzer. These words are displayed as ONEs and ZEROs and are selected with a pattern trigger and digital delay (Fig. 1).

The pattern, or data word, is selected by setting the 32 pattern-trigger switches to high, low, or off. The delay can be such that the display starts anywhere from 15 words before the pattern trigger, to 99,999 words after the trigger. Since the analyzer has a digital memory, data can be captured single-shot and displayed indefinitely.

Another useful analyzer feature is an output that can trigger a scope at any byte in a digital process. With the trigger, you can examine waveforms in detail.



```
0010 1010 0000 0111   0010 1010 1110 1010
0011 0111 0010 1010   0100 1101 1000 1000
0001 1100 0011 0111   0000 1101 1100 1110
0001 0100 0001 1100   0000 0000 1010 0110
1011 1011 0001 0100   0111 0010 0010 0100
0010 0101 1011 1011   0011 0011 1111 1000
0000 1010 0010 0101   0010 0011 0110 0110
0000 0100 0000 1010   0011 0100 1110 1000
1011 1011 0000 0100   0001 0100 0110 1110
0011 0101 1011 1011   0010 1001 1111 1110
0010 1010 0011 0101   0111 1101 1000 1110
0010 0111 0010 1010   0110 0001 0110 1110
0100 0010 0010 0111   1001 1101 1111 0111
0001 1001 0100 0010   0001 0011 1011 0010
0111 0010 0001 1001   0000 0111 0100 0010
0010 1010 0111 0010   0011 1011 1111 1110
```

1. A functional display of ONEs and ZEROs gives designers an overview of the operation of a digital system. Words are captured in the analyzer's memory.

Sequential operations cause problems

In developing a systematic process for turning on a μP system, remember that it's the data transactions on the various busses that determine a system's function. Consider the μP system of Fig. 2. If all of the elements in this system are wired up, plugged in and turned on, chaos will almost surely result. Any small wiring mistake or logic error can cause the system to run amok.

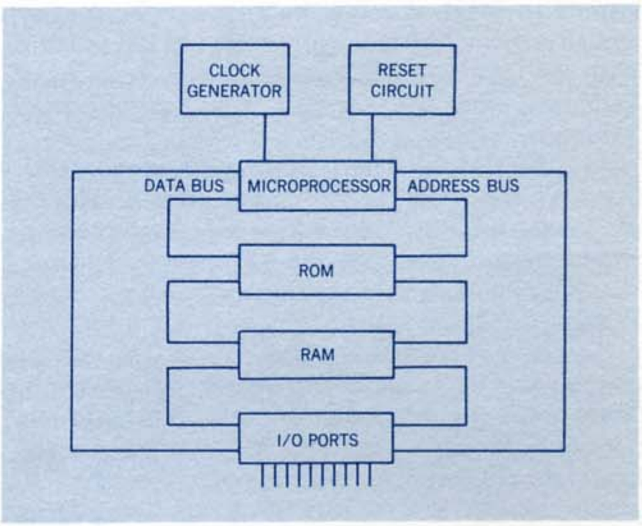
Such alarming behavior results because the entire system is one giant digital-feedback loop. The next value of the program counter depends on the current instruction, but the latter depends on the current value of the program counter.

Interchanging two address lines causes the instructions to be executed in an entirely random sequence. A small logic error, which allows two ROMs to respond simultaneously to an address, will cause the outputs of the ROMs to be wire-ANDed, again putting entirely random instructions on the data bus.

Similarly, data stored in memory—to be read back later for conditional branches—are determined by instructions executed long ago, and the instructions to be executed in the future depend upon those stored values. Current or post values of the I/O ports also participate in decisions that affect future operation.

The only reasonable approach to turning on such a system is to break the feedback network

W. A. Farnbach, Engineering Section Manager, Hewlett-Packard, 1900 Garden of The Gods Road, Colorado Springs, CO 80907.



2. Generalized microprocessor system contains fixed programs in ROMs and data in RAMs. Interaction between the memory and μP appears like a feedback loop.

into pieces, so that each piece can be turned on independently.

The first candidates for turn on are the clock generator and reset circuits. These relatively simple timing circuits are best tested with a scope for proper timing and waveshape. Once these circuits are within specs, the next step is to check the interaction between the μP and the ROM.

A crucial path: μP to ROM

The linkage between the ROM (or RAM) and the μP is the first to be established because with this link, the processor can be programmed to serve as a signal generator for testing the remaining blocks. The linkage is tested in three steps:

First, you must establish that the NOP (no operation) instruction is being transmitted correctly to the processor. Second, establish that the program addresses are transmitted correctly to the ROM. Finally, determine that the ROM interprets the program addresses correctly.

To perform the first step, plug in only the μP and put the NOP code on the data bus (Fig. 3a). In forcing the data bus to NOP, realize that many μPs will try to put data onto the bus during an operating cycle. If the data bus is simply wired to the NOP state, then the data-output buffers in the μP can be destroyed. You can avoid this problem in two ways.

Since the μP data inputs are usually high impedance, the data bus can be forced safely to NOP with large resistors. Or, it can be forced to NOP through a set of three-state gates. Connect the three-state control to the processor read/write line so that the gates are active only when the processor reads data.

This set-up will cause the program counter in the processor to increment. That is, the processor will execute a NOP, increment the program counter, execute the next NOP, and so on. You can easily measure the counting sequence on the address bus with the analyzer. Simply connect the 16 data inputs to the address bus at the ROM socket, and connect the clock input to the data-transfer processor clock.

The count sequence also can be easily verified. Just trigger the analyzer on 0000 , increment the delay generator through several values, and compare the count displayed on the state analyzer with the delay setting. Obviously, the count (in decimal) and the delay value will be equal if the ROM receives correct addresses.

In this way, you verify that the processor is executing NOPs and that the addresses are correctly transmitted to the ROM. If the addresses do not form a counting sequence, then an examination of the address pattern should quickly reveal if address lines are interchanged or are inactive, or the processor is executing an unexpected branch instruction.

If you suspect that the processor is not executing NOPs during the instruction read phase, then connect the state analyzer to the data bus directly at the processor.

Check waveforms at each change

At this point, it is also important to examine the waveforms on the busses and the control lines. Any incorrect timing, marginal voltage levels, noise or crosstalk should be eliminated before proceeding. In fact, you must do this every time you add a new block to the system. Any input or output hung onto a common line can cause a problem.

The analyzer's scope-trigger output is very useful, especially as more blocks are added. For example, to examine the waveforms on the data bus when the bus is driven by the RAM outputs, you need only trigger the analyzer on the RAM read address or on the address of the RAM read instruction, then trigger the scope with the analyzer's pattern-trigger output.

Although such testing may seem needlessly repetitive, it takes very little time if there is no problem, and saves a great deal of time, if there is one by pinpointing the troublesome block.

Next, plug in some ROMs with known stored information and connect only the address lines and chip-select logic (Fig. 3b). Since the instructions returning to the processor are still NOPs, the program counter will continue a simple count. This time, however, the ROMs will cycle through all possible addresses so that you can measure the ROM outputs with eight data inputs to the analyzer.

Keep 16 data inputs connected to the address bus, if possible. It isn't necessary to measure all possible values of ROM output, but you should check sufficiently to verify that the correct ROM is selected and that every ROM is addressed correctly.

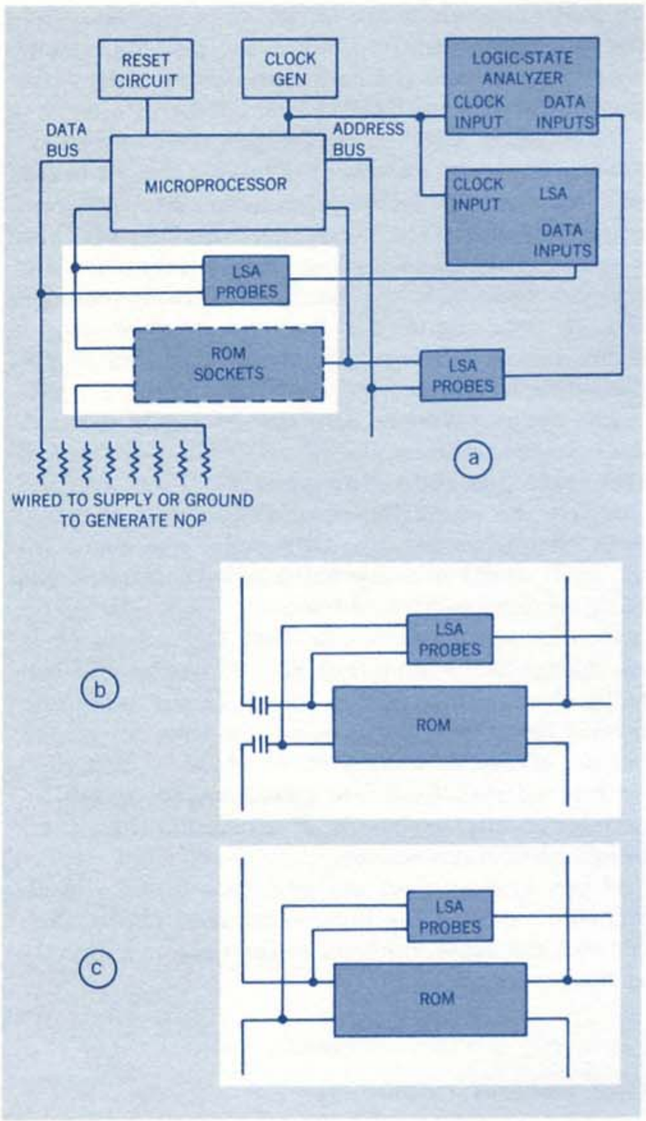
Since not all 65-k addresses are ordinarily allocated to ROM, it might be necessary to connect temporarily some pull-ups to the ROM outputs. With pull-ups, an address outside the allocated ROM addresses will generate a known data word (all highs).

You should also check some addresses outside those of the ROM to verify that the ROMs are off when they are not addressed. Remember to check the waveforms on the address bus and central lines, particularly on the control lines of the ROMs.

Finally, complete the processor-to-ROM link by removing any circuitry required to force the NOPs onto the μP and connecting the ROM outputs onto the data bus (Fig. 3c). A ROM containing a simple program, with several unconditional jumps, should be installed (Fig. 4). Verify operation of this program by monitoring the address bus with the analyzer.

The program includes RAM access and I/O instructions so that the RAM and I/O control cycles can be checked before the RAM and I/O devices are installed. The timing of these cycles is easily checked. Just use the analyzer to trigger a scope at the beginning of each RAM or I/O instruction.

It isn't necessary to monitor the data bus—unless there is a problem—because the sequence of program addresses is ample to verify proper execution of the program. Although only a very simple program is required to test the μP-to-ROM data link and the RAM and I/O control cycles, a more elaborate program can be used if desired.



3. To check transactions between the μP and ROM requires three steps. The first step verifies NOP transmission (a), the second checks ROM addressing (b) and the last step tests the completed link (c).

Debugging RAM and I/O

In no case, however, should any branches on RAM or I/O instructions be used at this point, as the RAM and I/O blocks have not yet been turned on and debugged. If enough ROMs are available, the test ROM, and any others used in the turn-on procedure, should be saved for future units.

The checkout of the μP-to-ROM data link is by far the most tedious. The reason is that this link must always be a feedback process. That is, each instruction depends on the address, and each address depends on the previous instruction.

The RAM and I/O blocks can be turned on much more directly, and in any order. If you choose the RAM first, you can connect it to the system in one operation.

With the RAM connected, run the ROM test program briefly to verify operation. Pay particular attention to the timing of the RAM control signals during the RAM read and write instructions. The usual cause of failure at this point is a shorted address or data line, two lines shorted together, or an unwanted RAM response.

Again, the analyzer will reveal quickly the location of the problem, and a scope triggered from the analyzer will show the nature of the problem. With the ROM program verified, now run a RAM test.

A RAM test program should write to every location in memory, then read each location back and verify the data. With an eight-bit-wide memory, watch out for a pitfall:

The eight bits of memory represent only 256 states. Conventional memories are usually much longer. This means that each possible data pattern must be written several times to fill the memory. If the same data are written into each block of 256 words, an error in any of the higher order addresses can be masked.

An extreme example of such masking is the case where all address lines (A8 to A15) are disconnected. Any simple perturbation of the 256-word pattern—such as shifting the pattern one word location in each block—will reveal the problem (Fig. 5).

For example, if you count from 0 to 255 in the first block, you should count from 1 to 255, then go back to ZERO in the next block; next, count 2 to 255, and go back to ZERO and ONE in the next block, and so on. The flowchart of an effective RAM-module test program is shown in Fig. 6. Remember, this test verifies that the memory system is working correctly—it does not check each cell of each memory location.

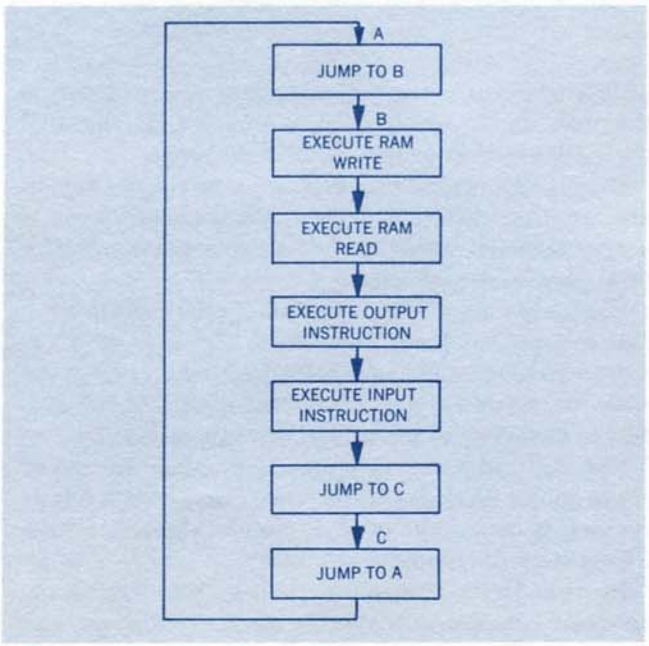
Again, if you design the program so that all locations are written and then read back, the analyzer quickly shows whether the data are correct. An oscilloscope triggered by an analyzer shows whether the waveforms are correct.

Although the I/O block is relatively easy to turn on, the discussion here is somewhat general since I/O structures vary more than other blocks from one μP system to another. The main point is to test the I/O ports before connection to peripheral devices, such as keyboards, displays, or circuits to be controlled. The first step is to put the ROM test program back in, and verify that the control timing is correct with the ports connected during the I/O instructions.

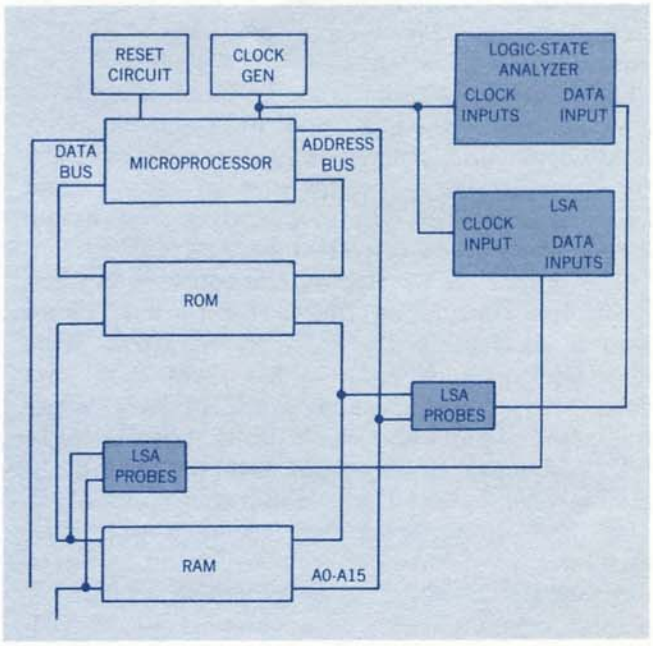
You can check the output ports easily with a simple program that first sets all the ports to ZERO, then sets each port in turn to ONE, and finally sets each port back to ZERO one at a time. When testing the output ports, connect the analyzer to one block at a time.

If sufficient data channels are available on the analyzer, connect these to the address bus as well (Fig. 7). The object of this exercise is to see if the output ports are connected in the proper order and can be set both high and low.

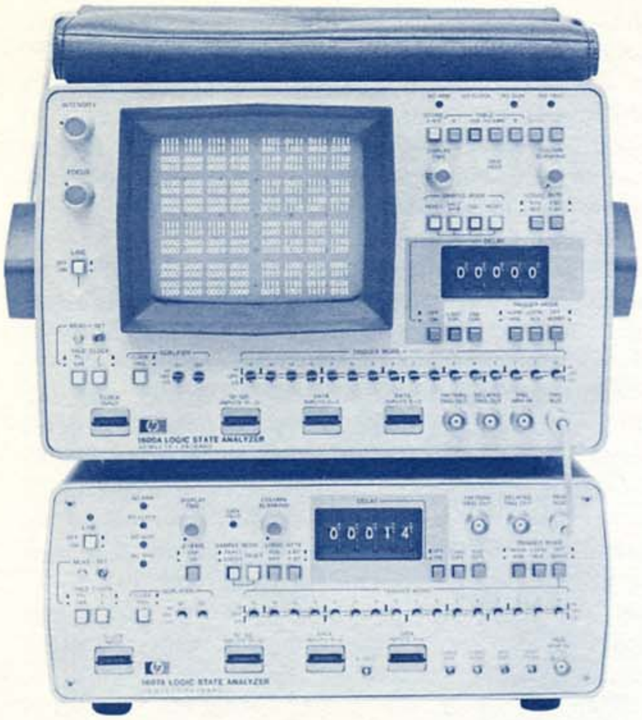
The input ports are similarly tested. The program should check for each input high, then for each input low.

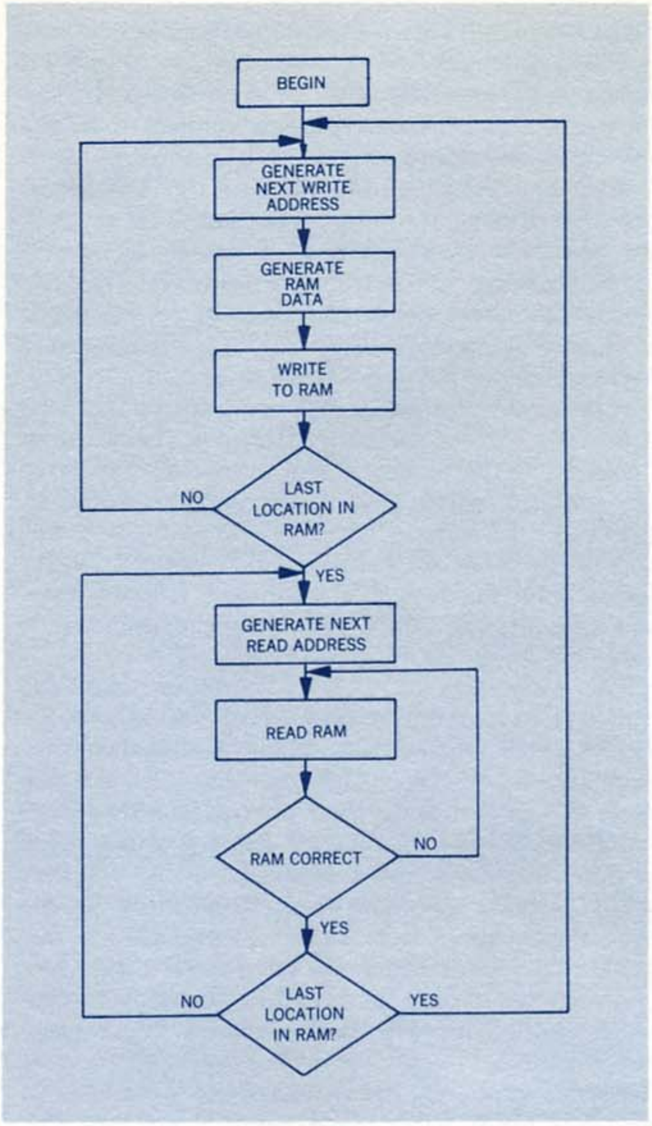


4. Verification program checks out the ROM-to-processor data link. Also checked are I/O control cycles.



5. Test set-up to turn on the system RAM verifies the writing and reading of each location in memory.


Logic state analyzer shows up to sixteen 32-bit words at a time. Data are put into memory when the instrument recognizes a selected word or are captured after a set delay.



6. **RAM-module test program** wrings out memory system operation, as monitored by the logic analyzer.

The test, of course, is performed by a program that loops until the input under test is forced to the desired state, then jumps to another loop (Fig. 8). A simple approach is to pull all of the inputs either high or low, whichever is easier, through a resistor.

Assuming you selected the "high" approach, write a program that has two loops: the first to test for a specific input low and the second for that input high. While the analyzer monitors the address bus and at least the one input under test, force the input low with a grounded wire. The analyzer will show which loop the processor is in, exactly when the input went low and—usually in a second pass—when the input went high.

Although this process may seem tedious, the time required to write the test programs must be spent only once. The programs will be invaluable at every phase of system development.

The process of developing the software is quite a bit like turning on the hardware. The major idea is to develop the software in pieces. This idea isn't new. Nobody in his right mind sits down, writes six-thousand words of code, plugs it in, and expects the whole thing to work right off. You must develop and test the coding in manageable bytes. Three alternatives are available: simulators, breakpoint registers and logic analyzers.

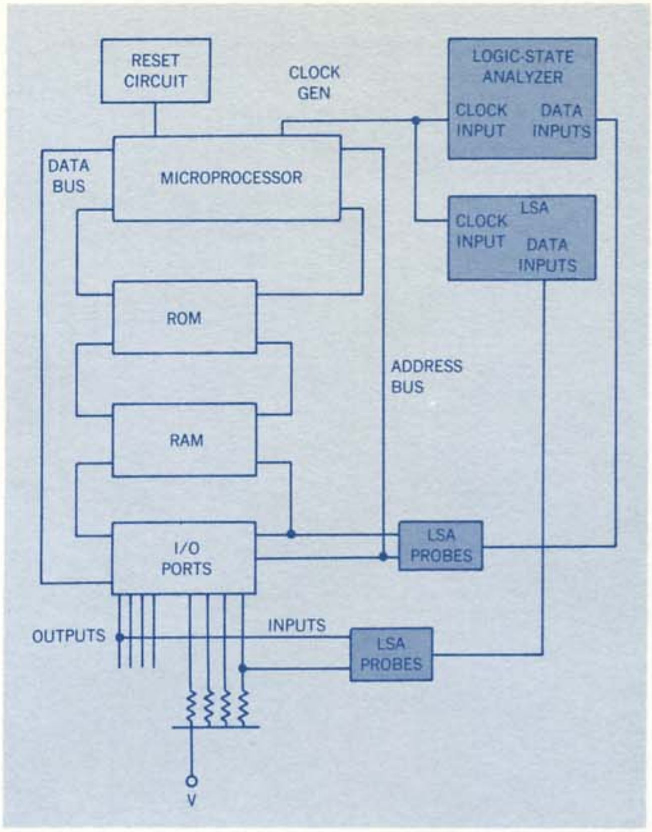
A simulator—either a development system or a large computer—can be a valuable aid in testing such complex algorithms as sorting routines or mathematical functions. But it is difficult to adequately simulate the software that performs the bulk of I/O operations—and it's at the I/O ports that major trouble usually develops.

Breakpoint registers and a single-step button are another way to follow the operation of a program. Such registers, or control panels, suffer from several drawbacks:

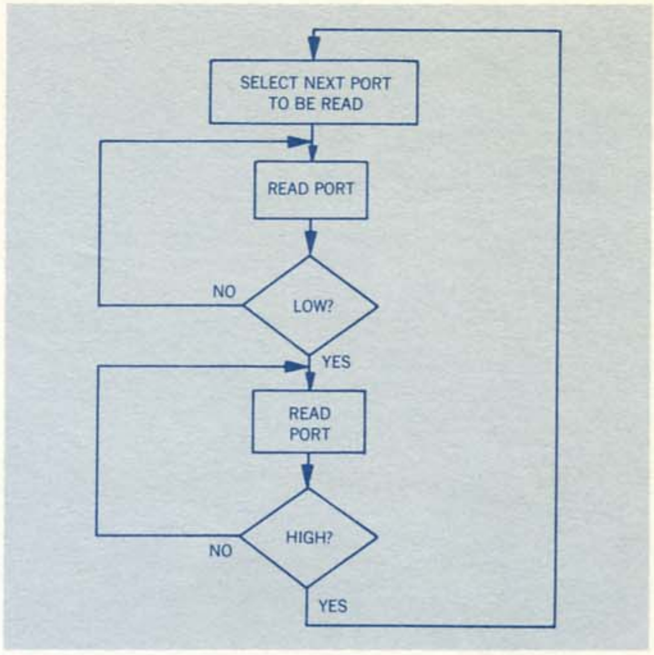
First, to build the control panel requires a fair amount of time and effort. Second—and far more serious—the operation of the processor must be slowed down by a factor of several million to observe the process at human speeds.

Not only does this great reduction in speed cause major changes in the operation of the whole system, it can make even a simple algorithm take a long time to complete.

In the third technique, using the logic-state analyzer, it doesn't really matter whether the software has been simulated beforehand or not. (Although, as mentioned before, if a simulator



7. Test the I/O ports before you connect the system's peripherals. First, verify control timing.



8. Input-port test program loops around to force the desired state, then jumps to another loop.

is available, it can be a help in developing some parts of the software.) One clear advantage of the analyzer approach is that the hardware and the software are debugged in parallel instead of in series. Another is that the analyzer can monitor the program flow in real time.

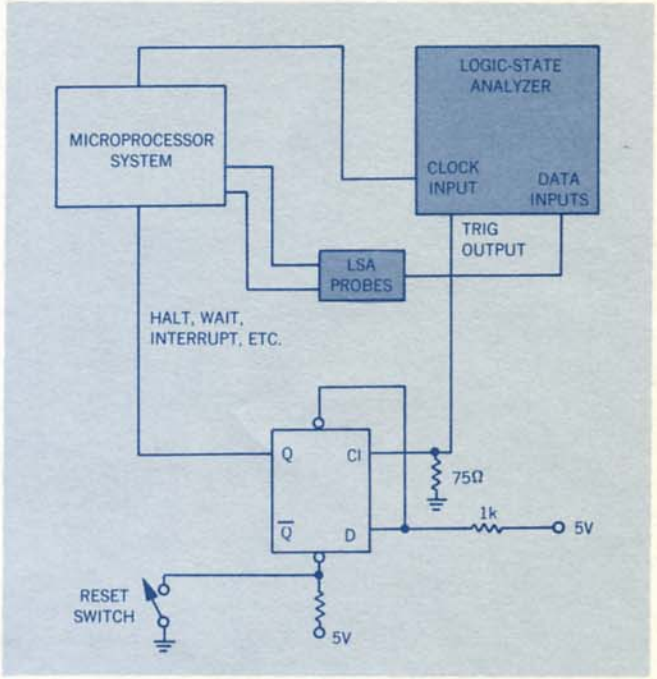
In debugging software, you use the analyzer in the same way as when debugging hardware. In fact, most of the hardware debugging techniques are simply a matter of monitoring the flow of a simple program, then fixing the hardware when the program does not work.

The process of debugging software, as it usually arises, is really more a problem of identifying a problem, deciding how the software and hardware contribute to the problem, then doing the fix.

In pinpointing whether software or hardware is the culprit, the logic analyzer excels. Once the hardware is checked out—from the lock and reset generators, to the I/O ports—the software can be loaded in small blocks and tried out.

Although you can debug the software in any order, several rules may be helpful. It is usually best to turn on the keyboard or other entry device first, then any display or output device. Next, turn on the hardware and software together.

Note that the logic-state analyzer can serve as a breakpoint register. Connect its trigger output to a flip-flop and use the flip-flop output as the break signal (Fig. 9).



9. Connect the analyzer's trigger output to a flip-flop, and the instrument can halt or interrupt μP action.

Bibliography

Farnbach, W. A., "Logic State Analyzers—a New Instrument for Analyzing Sequential Digital Processes," *IEEE Transactions on Instrumentation and Measurement*, Vol. IM 24, No. 4, December, 1975, pp. 353-356.

Farnbach, W. A., "Troubleshooting in the Data Domain Is Simplified by Logic Analyzers," *Electronics*, May 15, 1975, pp. 103-105.

House, C. H. "Engineering in the Data Domain Calls for a New Kind of Digital Instrument," *Electronics*, May 1, 1975, pp. 75-81.

Small, C. T. and Morrill, J. S. Jr., "The Logic State Analyzer, A Viewing Port for the Data Domain," *Hewlett-Packard Journal*, August, 1975, pp. 2-10.

HEWLETT hp PACKARD

For More Information, Call Your Local HP Sales Office or, in U.S., East (301) 948-6370. Midwest (312) 677-0400. South (404) 434-4000. West (213) 877-1282. Or, Write: Hewlett-Packard, 1501 Page Mill Road, Palo Alto, California 94304. In Europe, Post Office Box 85, CH-1217 Meyrin 2, Geneva, Switzerland. In Japan, YHP, 1-59-1, Yoyogi, Shibuya-Ku, Tokyo, 151.

5952-2036

PRINTED IN U.S.A.