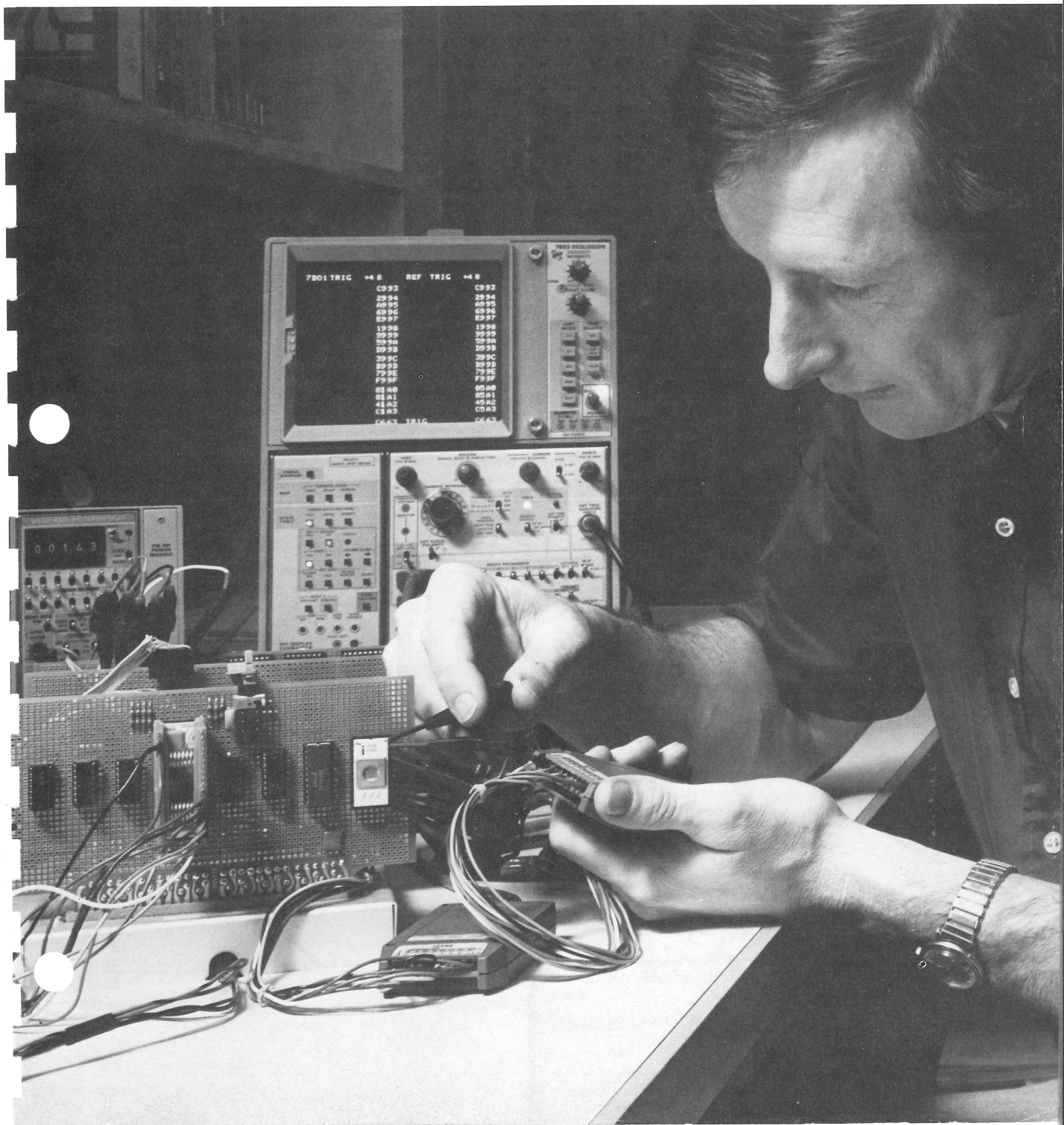


TROUBLESHOOTING A MICROPROCESSOR

Logic Analyzer Application Note # 57K1.0



TROUBLESHOOTING A MICROPROCESSOR

WHY A LOGIC ANALYZER INSTEAD OF A SCOPE?

Maybe you're just getting started with microprocessors, or maybe you've worked with them from the beginning. Either way, you've probably discovered that designing and troubleshooting complex digital circuitry requires a special kind of tool . . . a tool with some capabilities the oscilloscope doesn't offer. The ability to acquire, store, and display 16 or more channels of data at one time; to capture data preceding a trigger; to trigger on a desired word or pattern; to format data in a map or state table as well as in the traditional timing diagram and decode that data from binary to hexadecimal or octal notation; all these are necessary (or, at the least, very useful) in the data domain.

The oscilloscope is still essential for detailed real-time analysis of electrical problems, but the *logic analyzer*, its counterpart in the data domain, meets these new requirements.

This application note takes you through a typical microprocessor troubleshooting sequence: from mapping to state tables to timing diagrams. Every step can be performed with the 7D01F logic analysis package described here.

The 7D01F features 16-channel data acquisition and word recognition, 4k formattable memory, pre, center, or post trigger, synchronous and asynchronous timing. Data may be mapped, formatted as state tables, or presented in timing diagram format; all three formats feature readout in binary, hex, or octal notation.

The addition of the WR501 Word Recognizer provides up to 32 channels of word or pattern recognition plus digital delay capability.

WHAT IS MAPPING?

Mapping is a powerful analytical technique. The entire contents of the 7D01F memory are displayed at once, and a unique dot pattern is generated. This unique pattern greatly simplifies such procedures as checking program flow or obtaining a software signature.

How is the data mapped? In the case of 16 bit wide words, the inputs from channels 8 to 15, representing the most significant byte, are inputted to a D/A converter which drives the vertical deflection circuitry of the crt; the inputs from channels 0 to 7, representing the least significant byte, are inputted to another D/A converter which drives the horizontal deflection circuitry. Thus each combination of 1s and 0s in a word deflects the beam to a particular position on the crt.

With 16 bits, $2^{16} = 65,536$ unique memory locations are displayed in a 256×256 matrix. What happens is that we take the 254 words (each 16 bits wide) stored in the 7D01F memory and display all 254 of them as dots at the position defined by the combination of 1s and 0s that makes up each word. If some of the 254 words are identical, then, they will be displayed in the same spot, making that location brighter than the others (See figure 1).

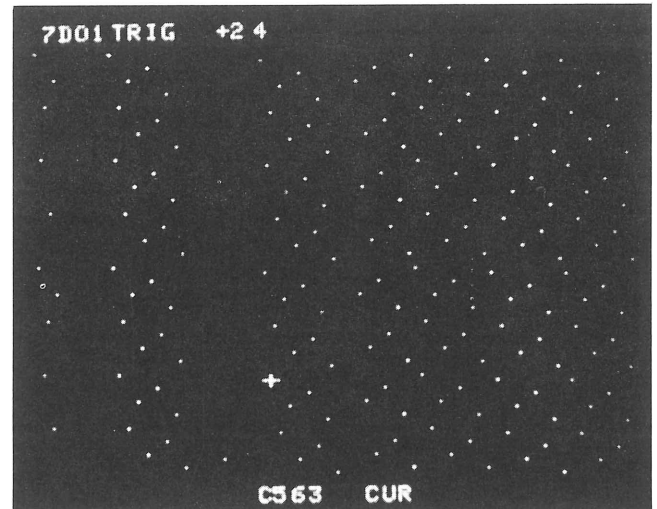


Figure 1.

WHY IS MAPPING HELPFUL?

Mapping enables you to quickly look at all 254 words and compare their relative numerical values. Thus, in a microprocessor application, you might want to analyze an address bus that is 16 bits wide. The pattern of 1s and 0s on that bus tells us which device (and which location in that device) takes data from or puts data onto the data bus. So if we know the address, we know what device the microprocessor is operating on.

The total number of addresses available is broken into blocks for the supporting memory and I/O. For example, hexadecimal addresses 0000 to 00FF might be reserved for a 256×8 bit RAM (where the data that the microprocessor works on is kept). Addresses 0100₁₆ to 0900₁₆ might be assigned to a $2k \times 8$ bit PROM, where the instructions which make up a microprocessor's program are stored. And addresses FF00₁₆ and FEFC₁₆ might be the addresses of two 8 bit wide I/O ports that communicate with a terminal. If the microprocessor were talking with the terminal, you would expect to see some dots at the locations defined by FF00₁₆ and FEFC₁₆, as well as some in the ROM address space (for instructions) and some in the RAM address space (where the microprocessor stored what the terminal sent to it). (See figure 2).

RAM addresses 0000₁₆-00FF₁₆

ROM addresses accessed to
get program steps stored
there

I/O ports

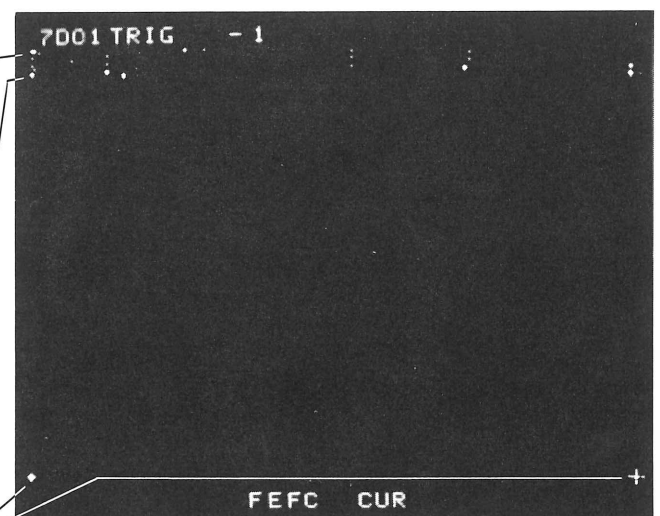


Figure 2.

Now, rather than tediously going through all 254 words stored in the 7D01F and looking for FF00₁₆ or FEFC₁₆, you could easily see at a glance whether the microprocessor ever went to the I/O port during a program. *That* is the power of mapping.

A larger microprocessor system might have much more program memory. By mapping the address line activity, you would quickly get an idea of which routines the microprocessor was performing. For example, a block data transfer routine might be in this area:

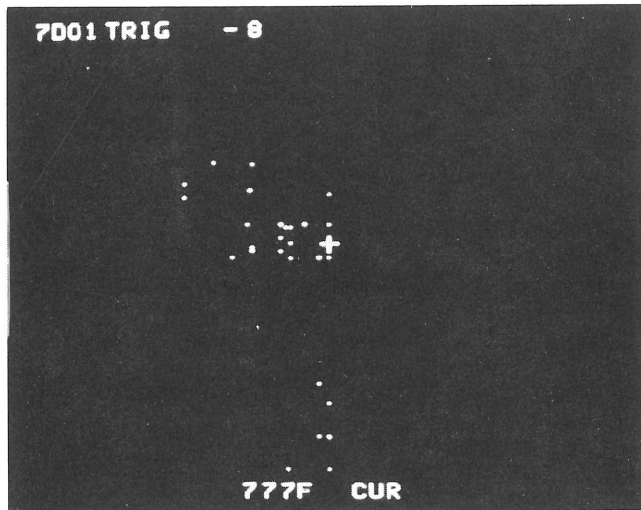


Figure 3.

But suppose you find later that the map looks like this:

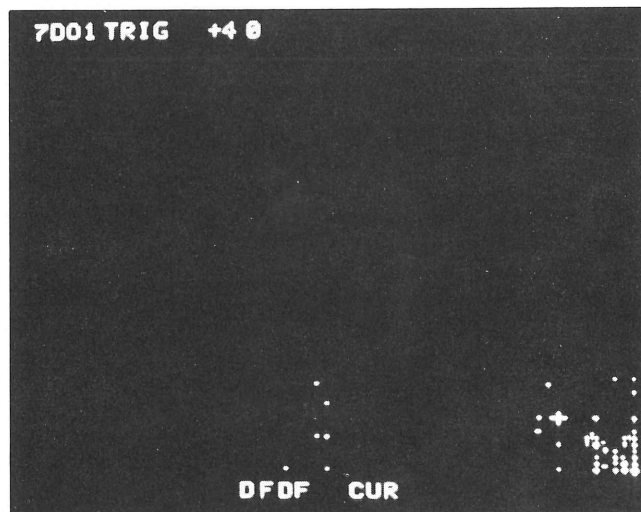


Figure 4.

Obviously, the microprocessor is doing something other than what you had in mind.

By locating the address of one of these invalid points with the cursor (+), you could quickly set up a trigger on the built-in 7D01F word recognizer. Then you could restart the microprocessor and wait for the 7D01F to trigger. When it did, you would have a picture of how the microprocessor got to those invalid addresses. By using the cursor, you could then track the addresses sequentially and see just what location the address jumped from (See figure 5).

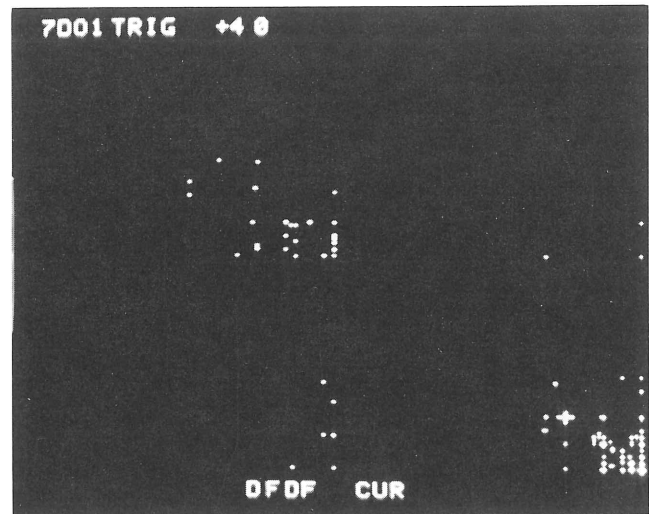


Figure 5.

WHAT GOOD IS STATE TABLE INFORMATION?

State table displays are important aids for debugging software. Once you've located a problem area with mapping, you can go on to the next step: locating the faulty bit or bits in the program itself.

In a binary system, a high voltage level might mean "1" and a low voltage level might mean "0". The parallel combination of 1s and 0s on a group of lines at one point in time would represent a word or state, e.g. 1111111011111100. This is a 16-bit word displayed in binary.

Here is the same number displayed in hexadecimal, where groups of 4 bits have been decoded into digits ranging from 0 to F, beginning at the right:

FEFC

You might recognize FEFC as the address of an I/O port (figure 2). Would you recognize it as quickly in binary form? This is the value of hex (or octal) decoding: masses of data are reduced to an easy-to-use form. The state table display, then, is just another way of looking at data the 7D01F has stored.

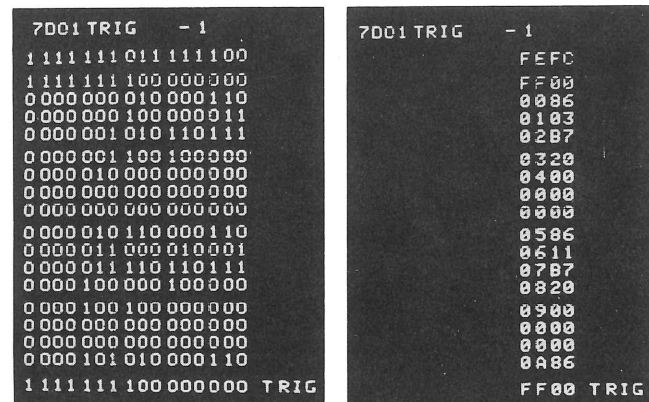


Figure 6.

But what about the state data itself? How is it different from voltage waveforms? Like mapped data, state tables differ from timing diagrams in that the data is stored using the same clock as the microprocessor system, or synchronously. The microproces-

sor system ignores the voltages present on a bus between active clock edges. In other words, the voltages on a bus are sampled by the microprocessor only at certain times, and it is to our advantage to use the same samples the microprocessor uses so that we store only what the microprocessor "sees".

As an example, let's take the map of address lines from the previous section on mapping. Now that you've stored the incorrect block of addresses, you switch over to the state table mode, which shows you the address word at the cursor location and the next 16 words stored—17 of 254 words total that have been captured. Then, by using the cursor control, you can locate the illegal address area; in this case, DFD7, toward the bottom of the table (See figure 7).

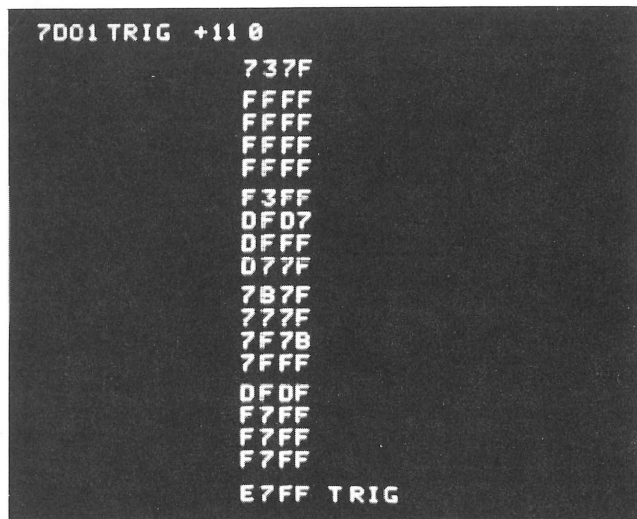


Figure 7.

Now it's an easy matter to see what devices the microprocessor was accessing just before it went wrong. If this were in the program ROM, you could check your listing of its contents and see exactly what the microprocessor was supposed to be doing.

WORD RECOGNITION WITH THE WR501

With the WR501 as part of the 7D01F package, you have 34-channel word or pattern recognition with 16 channels stored and displayed. The digital delay feature allows you to delay from the word recognizer inputs by words or clock pulses. With word recognition and digital delay both, it's possible to word-recognize on word or pattern A, then on B, then to delay; or to word-recognize on A, delay for a certain count, then recognize on B.

WHAT DOES WORD RECOGNITION DO FOR YOU?

Let's continue our example from the state table section. You now know that the microprocessor was accessing the PROM (program storage area) just before it went bad. By attaching the address bus probes to the WR501, setting up the WR501 Word Recognizer for the first illegal address, and attaching the 7D01F to the data bus, you can record data bus activity around the point of failure.

An examination of the state tables showed that the microprocessor received the code for a jump instruction and, in the next two lines, received DFD7 as the address it should go to. However, DFD7 is the first address in the illegal block of code. It appears that the PROM has been incorrectly programmed.

The WR501 has triggered the 7D01F, allowing you to store the area of operation you were looking for on the data bus, and so to locate the fault.

THE 7D01F REFERENCE MEMORY

The 4k 7D01F reference memory is separate from the working memory. When a block of data has been stored in the 7D01F and is found to be correct by your analysis, it can be "set aside" as a reference. You can then specify that data gathered on subsequent storage be ignored if it matches a 17-word table you select with the cursor, or if it matches the entire 254-word reference memory. The 7D01F will count the number of correct matches, generate an external TTL reset pulse, and reset itself until a mismatch occurs. Then it will hold the erroneous data and intensify it in the display, so you know immediately *where* it is and *what* it is.

Suppose the microprocessor must perform a given sequence upon restart; but suppose also that it fails occasionally. First you capture a good sequence restart, designate that as your reference, and, using the 7D01F external reset pulse to restart the microprocessor, you activate the Reset If mode. Then you go home for the night. Upon your return in the morning, you'll find the 7D01F holding the record of the 36th restart sequence, which is incorrect.

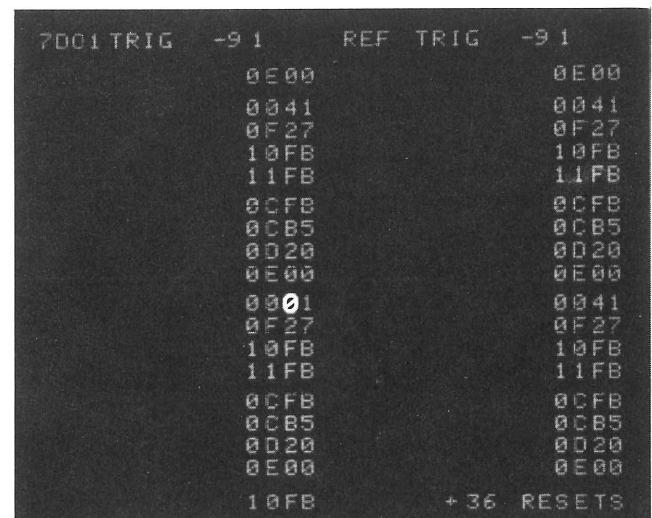


Figure 8.

The sequence is analyzed and the problem solved. As you can see from this example, the reference memory with intensified exclusive-OR comparison can be a real time-saver.

WHAT ABOUT THE TIMING DISPLAY?

Timing diagrams may be generated by using the 7D01F's internal time base. In this manner, 16 channels may be sampled at up to 50-ns intervals, 8 channels may be sampled at up to 20-ns intervals, or 4 channels may be sampled at up to 10-ns intervals. The channels can then be displayed as pseudo-waveforms in the familiar timing diagram format.

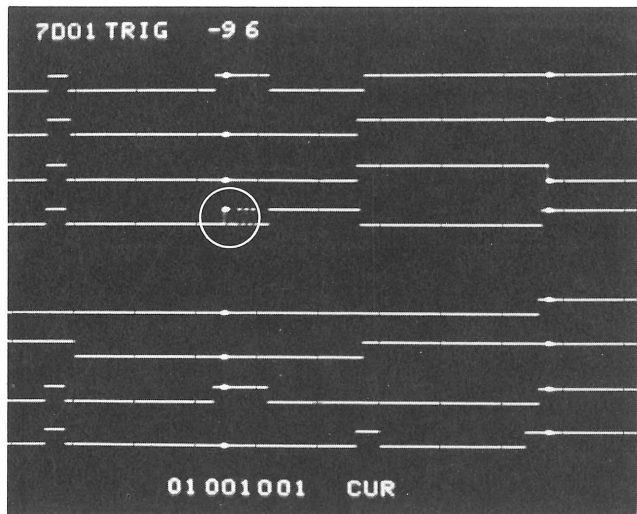



Figure 9.

How is this useful? Suppose, in the failure example described in the state table section, that the fault has been traced to an I/O port instead of to a PROM. Incorrect data is being presented to the microprocessor by the I/O port. Is the incorrect data coming from the I/O port or the hardware on the other side?

By using the 7D01F internal time base as an asynchronous clock, the keyboard-generated waveforms on the "outside world" side of the I/O port can be examined in detail. (And they can *only* be examined asynchronously.) In our example, the I/O port is found to be perfectly okay. The trouble is traced to a faulty pullup resistor on the switch input to the keyboard encoder . . . to a race condition in the keyboard debounce circuitry . . . to a faulty Schmitt trigger . . . (circled area figure 9) or to any number of internal or external problems.

In conclusion, then, the 7D01F logic analyzer package provides both synchronous and asynchronous testing capabilities. Use map and state table formats for checking the internally clocked, or synchronous, bus structure of the microprocessor system itself. Use the timing diagram for troubleshooting the microprocessor's only link with the outside world, its supporting digital circuitry. Either way, the 7D01F gets you through your test procedure.

Copyright © 1977, Tektronix, Inc. All rights reserved. Printed in U.S.A. Foreign and U.S.A. Products of Tektronix, Inc. are covered by Foreign and U.S.A. Patents and/or Patents Pending. Information in this publication supersedes all previously published material. Specification and price change privileges reserved. TEKTRONIX, TEK, SCOPE-MOBILE, TELEQUIPMENT, and  are registered trademarks of Tektronix, Inc. P.O. Box 500, Beaverton, Oregon 97077. Phone: (Area Code 503) 644-0161, TWX: 910-467-8708, Cable: TEKTRONIX. Overseas Distributors in over 50 Countries.