A "Trick" with RUN UUT

by Randy Mather

Here is a revelation of yet another Fluke 9010A "trick" that may not be known to many users.

As the pod manuals have pointed out, the pod cannot operate with the Unit Under Test (UUT) and the mainframe at the same time. The microprocessor (μP) in the pod spends most of its time monitoring the mainframe waiting for a command from the Troubleshooter. During this "overhead" time the pod is sending out read operations at the reset address so that any UUT refresh activity may take place. This overhead time occurs between READ or WRITE commands. (Remember that all of the 9010A functions like LEARN, RAMP, DTOG, etc. are comprised of the basic READ and WRITE functions).

In designing tests for a μP-based UUT, this overhead time may make a 9010A built-in test or test sequence ineffective for some applications. For example, you may feel that the RAM

SHORT test takes too long to execute for a 64K-byte section of RAM. In this case, you might test a smaller section of RAM with the built-in test and then download a UUT program into the tested portion of RAM. This UUT program would then test the rest of RAM using the RUN UUT function. Interrupt testing is another situation in which you might have to resort to executing a UUT program.

When you use the RUN UUT function, the μP in the pod takes commands only from the UUT and effectively cuts off all communication with the mainframe. The mainframe has no way of knowing when the UUT program has finished. This lack of communication, between the pod and mainframe, can present problems when the RUN UUT command is used within a 9010A program. The following program is a typical example of this RUN UUT problem.

```
PROGRAM XX
•
•
WRITE @ 5056 = 34
RUN UUT @ 5000
READ @ 5057
•
•
```

The WRITE statement may be the last in a string of WRITE commands that was used to download a UUT program into RAM. The first instruction of the UUT program is at address 5000, so the RUN UUT command must be at that address. For this example, when the UUT program is run it will test the remainder of RAM and then write the results of the test into address 5057. The 9010A program above will read address 5057 and branch according to the results of the test.

The problem with this program is that the READ @ 5057 will stop RUN UUT approximately 10 milliseconds after it starts. After the RUN UUT command, the mainframe, not knowing if the UUT program is finished or not, blindly moves on to the next step. The READ @ 5057, being a pod-related statement, stops RUN UUT. This is fine if the run time of the UUT program is less than 10 milliseconds. If it is longer, the READ operation interrupts the RUN UUT execution, stopping the UUT program.

One way to get around this problem is to place a STOP just after the RUN UUT command and have the operator press CONTinue when he or she feels that enough time has elapsed for the UUT program to finish.

However, the STOP method is somewhat clumsy and leaves too much to chance with the operator interaction. You can devise a program that senses when the UUT program has stopped. The "trick" to this is based on the fact that even though the μP within the pod is executing UUT machine code, the μP within the 9010A mainframe continues executing the 9010A high-level language. As long as the mainframe program does not require any pod action, the UUT program

will continue to run uninterrupted.

One way to use this trick is to have the 9010A execute a time delay program that is just a little bit longer than the run time of the UUT program, as in the following example.

```
PROGRAM XX
•
•
WRITE @ 5056 = 34
RUN UUT @ 5000
EXECUTE PROGRAM YY
READ @ 5057
•
•
PROGRAM YY
REG 1 = 45
LABEL 1
DEC REG 1
IF REG 1 > 0 GOTO 1
```

The mainframe will set the UUT program in operation with the RUN UUT command and then execute program YY, which decrements register 1 down to zero. Then the mainframe will execute the READ @ 5057 statement. The initialization value for register 1 in program YY will have to be adjusted so that program YY runs just a little longer than the UUT program. In this example the 9010A sets the time delay (instead of the operator, as in the previous example).

This program may work fine for those UUT programs that run a specified length of time, but would be totally unsatisfactory when the UUT program is controlled by asynchronous events, like interrupts, etc. The next example shows how to take care of this problem.

The next method, using our RUN UUT trick, requires some changes in both the 9010A program and the UUT program. It requires finding a node on the UUT that is controlled by the μP but is not used by the UUT program (i.e., unused decoder outputs, unassigned address lines, peripheral outputs, etc.). Once you find such a node, design your UUT test program so that the last thing it does is set this node to a specified level. If the node is not latched, you will have to design your UUT program to continually loop while sending stimulus to that node.

In this example, the 9010A program will be designed to continually look at the probe level, instead of executing a time delay program.

```
PROGRAM XX
•
•
WRITE @ 5056 = 34
DPY-PROBE PIN x U y
STOP
RUN UUT @ 5000
EXECUTE PROGRAM ZZ
READ @ 5057
•
•

PROGRAM ZZ
SYNC FREE RUN
READ PROBE
LABEL 1
READ PROBE
IF REG 0 AND 4000000 = 0 GOTO 1
```

After placing the probe on the desired node, according to the displayed message, the operator presses CONT and the pod sets the UUT program in motion, in response to the RUN UUT command. Program ZZ is then executed, looking at the node where the probe is placed and, in this case, waiting until the LOW bit in register 0 is set (reg 0 = 4000000). A mask of 1000000 is used if you want to check for a HIGH level. Once a LOW level is detected, then program ZZ will complete, and execution will return to program XX, executing the READ statement.

One important thing about program ZZ is that you *must* sync the probe to FREE-RUN. If you are synchronized to any other mode (i.e., ADDRESS or DATA) the 9010A will not enable the probe. In RUN UUT mode, no sync signal is sent to the probe from the pod. However, in FREE-RUN, 1-kHz pulse is sent to the probe, allowing it to see any level change at the node. (It may be necessary to insert another SYNC command after the EXECUTE PROGRAM ZZ statement to put the probe back into the previous sync mode.)

One other method of using the RUN UUT trick is to look at the "character received" bit in the RS-232 status word. This can be used if the UUT has an RS-232 port and your 9010A has the RS-232 option. In this case, the last thing the UUT program must do is send a character out through the UUT's RS-232 port to the 9010A. With the two RS-232 ports tied together, the 9010A program will sense that a character has been received and then branch back to the READ instruction, as before.

This trick in using the RUN UUT instruction effectively gives the user the power of two processors working at once for added flexibility and performance. The possible uses are limited only by your imagination.

Find that bus short

by Howard Kaplan

When troubleshooting μP-based systems with a 9000-Series Micro-System Troubleshooter, there will be a class of errors that point to a device or devices on a common bus having a low-impedance fault (a short) to Vcc, to ground, or between devices. With this type of fault (especially among soldered-in devices), it is extremely difficult to locate the actual failed component.

A technique is needed to locate the offending component or trace, without having to desolder IC s (and run the risk of destroying the PCB). This article describes such a technique that uses the diagnostic capability of the Troubleshooter along with its logic probe and a Hewlett Packard Model 547A Current Tracer.

The problem can be divided into two basic parts: locating the faulty node or nodes using the Troubleshooter, and isolating the failed component using current-tracing techniques.

Locating the Faulty Node

The Troubleshooter will generally report two classes of error messages, depending on whether the fault is directly on the μP bus or is beyond address and data buffers present in the system.

For errors that are directly on the μP bus (drivability errors), BUS TEST should always be used to locate the fault. The following are several types of BUS TEST error messages with corresponding fault descriptions:

MESSAGE	FAULT
ADDR BIT 0 TIED LOW - LOOP?	A0 tied low
ADDR BIT 1 TIED HIGH - LOOP?	A1 tied high
ADDR BITS 3 AND 4 TIED - LOOP?	A3 and A4 tied
DATA BIT 1 TIED LOW - LOOP?	D1 tied low
DATA BIT 2 TIED HIGH - LOOP?	D2 tied high
DATA BITS 3 AND 4 TIED - LOOP?	D3 and D4 tied

Once an error has been found, the procedure for locating the faulty component consists of LOOPing on the indicated error and applying the current-tracing technique (described later) to the indicated faulty node or nodes.

For errors that are beyond the address and data buffers, the problem of diagnosing the error becomes more complex. In general, a good method is to perform a RAM SHORT test over the RAM address space of the UUT until a fault is identified by the Troubleshooter. Then look at the Troubleshooter display to see if the MORE indicator is on and, if it is, press the MORE key to display the additional information. Using this information, decide which node is at fault on the UUT and then use the current-tracing technique (described later).

Troubleshooting Examples

The following examples illustrate the technique for locating the faulty node in a system that has RAM at 8C00-8FFF (hex).

Example 1: Data bits D3 and D4 shorted together.

TEST OR KEY STROKE	DISPLAY MESSAGE
RAM SHORT @ 8C00 - 8FFF	RAM BITS 3 AND 4 TIED - LOOP?
ENTER/YES	RAM BITS 3 AND 4 TIED

Then locate the source of D3 or D4 and apply the actual current-tracing technique, moving away from the source of either D3 or D4.

Example 2: Address bits A3 and A4 shorted together.

TEST or KEY STROKE	DISPLAY MESSAGE
RAM SHORT @ 8C00 - 8FFF	RAM DCD ERR @ 8C00 BIT 3 - LOOP?
CLEAR/NO	RAM DCD ERR @ 8C00 BIT 4 - LOOP?
ENTER/YES	RAM DCD ERR @ 8C00 BIT 4

Then locate the source of A3 or A4 and apply the current-tracing technique, moving away from the source of A3 or A4.

Isolating the Failed Component (Current-Tracing Technique)

Once a suspect node has been identified, the procedure to isolate the failed component can be broken into the following steps:

1) Identify the source of the problem logic signal. If there are multiple possibilities, arbitrarily choose one as the source.

2) Set the Troubleshooter to the ADDRESS SYNC or DATA SYNC mode, depending on the function of the signal being traced. The Troubleshooter should be LOOPing on the error.
3) Connect the Troubleshooter probe as close as possible to the chosen source of the logic signal. Set the Troubleshooter to pulse HIGH and LOW.
4) With the HP 547A Current Tracer as close as possible to the Troubleshooter logic probe, adjust its threshold control until the indicator light is at approximately ¾ intensity.
5) Gradually move the Current Tracer away from the Troubleshooter probe, following the PCB traces. If this is not possible, use other IC pins on the node as test points. Do this until a point is reached where the Current Tracer light goes out. At this point you have gone beyond the point of the low-impedance fault and need to back up to the point where the light just comes on. Continue in this manner, always stopping at points where the light goes out (no fault) and back-tracking to the point where the light is on (fault). The point at which you reach a dead end with the light on, or a transition from on to off with no other possible paths to trace, is the location of the short.

Current Tracing Example

The following example illustrates the current tracing technique. Assume we have a μP system with an unbuffered data bus that has multiple devices capable of driving the bus (see figure 2). One of the devices is driving the bus when it shouldn't be and we are attempting to locate the actual device at fault.
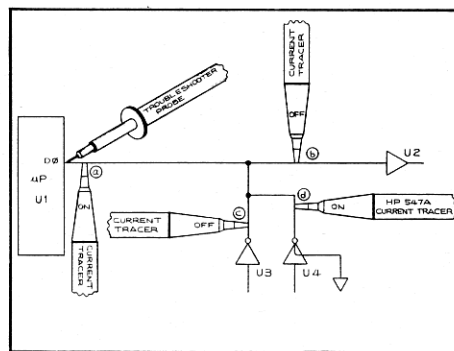

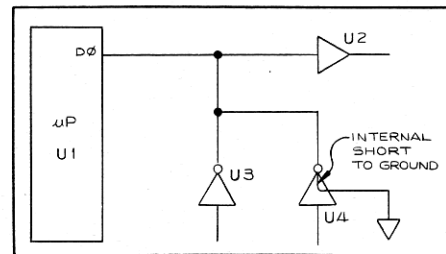**Figure 1. Current-Tracing Procedure.**

Figure 2. Fault for Example.

1) Perform a BUS TEST with the Troubleshooter. The result indicates we have D0 TIED LOW. Locate the source of D0 (hex bit 01).
2) Set the Troubleshooter to DATA SYNC mode. Restart the BUS TEST and LOOP on the D0 TIED LOW error message.
3) Connect the Troubleshooter probe to the D0 line as close as possible to the μP socket or to the 9000A-7201 Probeable Socket (PN648907). Set the Trouble-shooter to pulse HIGH and LOW.
4) Set up the HP 547A Current Tracer and adjust the threshold control for ¾ intensity with the Current Tracer at the Troubleshooter probe location (see figures 1(a) and 3).
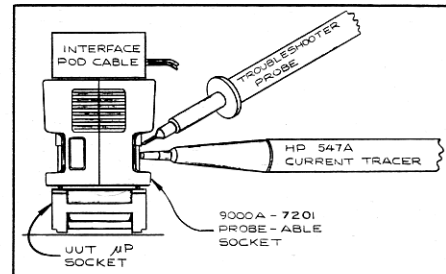

Figure 3. Current Tracer Adjustment.

5) Move the Current Tracer along the D0 trace (see figure 1) until the light goes out. In this case, first at U2 (b), which indicates that the fault is not in that direction. Proceed tracing back at the junction going toward U3 and U4, noting that the light comes back on again. Continue on to U3 (c), and note that the light goes off, indicating that U3 is not at fault. Go back to the U3/U4 junction and continue toward U4. You will then discover that the light stays on all the way to the location of U4 (d). The conclusion is that U4 is drawing the fault current.

At this time, one should not immediately assume that U4 is at fault. The inputs to U4 should be verified for correct conditions to decide if the input circuitry is at fault or if there really is an internal short in U4.

In conclusion, the technique outlined above combines the powerful diagnostics capabilities of the 9000-Series Micro-System Troubleshooter with the Hewlett Packard 547A Current Tracer into an easy-to-use method for isolating low-impedance faults, without having to desolder suspected components.

Testing non-μP digital assemblies

by Mike Petterson

Have you ever wanted to check out a digital assembly with your 9000-Series Micro-System Troubleshooter, but found that the assembly didn't use a microprocessor? This article outlines a way to put the power and convenience of the Troubleshooter to work on your non-μP digital assemblies.

Digital Assemblies

For the purposes of this article, a digital assembly is any piece of equipment that can be adequately tested by stimulating and reading only digital logic levels. This would include logic boards, display boards, and cable assemblies (because digital levels are sufficient to check continuity and proper connections of a cable). In the remainder of this article, the term Unit Under Test (UUT) will refer to a digital assembly.

The Pod Support Fixture

The first step is to select an interface pod to use. Generally, any pod can be used, though some are more easily interfaced than others. The single power supply pods (e.g., Z80, 8085) are usually good choices because the power requirements are simple. If use of the test fixture won't be frequent enough to justify the purchase of another Troubleshooter interface pod, then availability is another good consideration.

The next step is to fulfill the basic requirements of the interface pod. This is done by constructing a simple Pod Support Fixture that supplies a clock signal, proper power supply voltages, and proper conditioning for status lines (by pulling them to a benign state). The Pod Support Fixture also provides a socket and physical support for the pod cable.

The following are schematic diagrams for Pod Support Fixtures for a few pods.
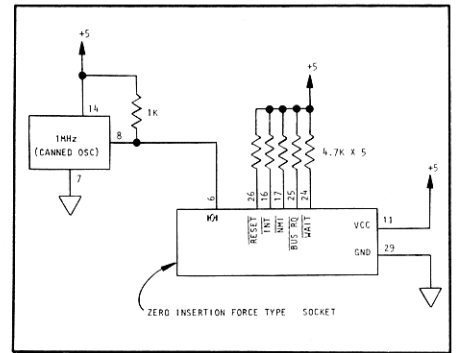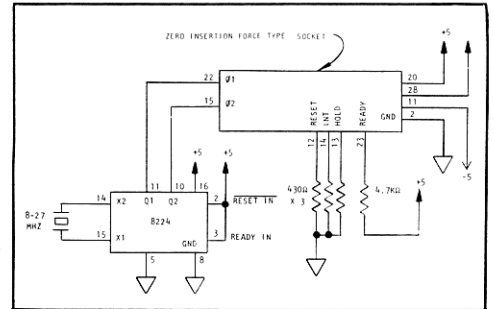

Figure 1. Z80 Pod Support Fixture

Figure 2. 8080 Pod Support Fixture

U1, U3–U10, U19   74LS541
U2, U20   74LS138
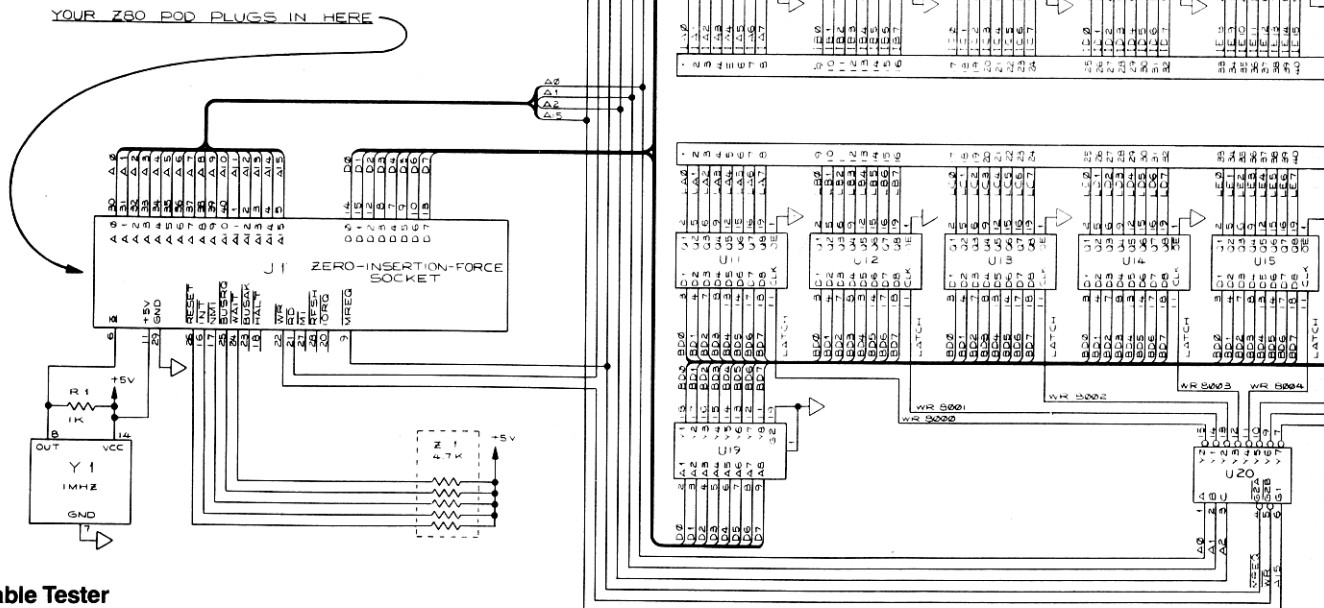U11–U18   74LS374
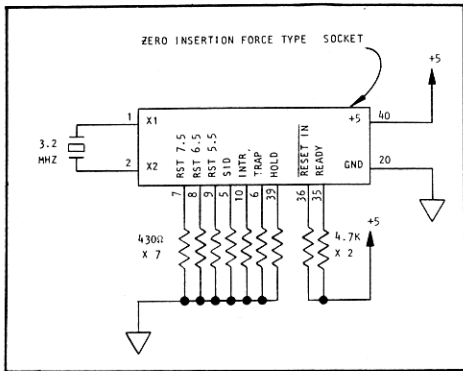

Figure 4. Cable Tester

Figure 3. 8085 Pod Support Fixture

Testing the Digital Assembly

Once the Pod Support Fixture has been constructed, the address, data, and control lines are available for use in testing the UUT. In applying the Troubleshooter to this type of testing problem, the object is to set up a bit pattern at the inputs of the UUT and then read back the resulting outputs from the UUT. The result can then be compared to an expected value and a pass/fail decision can be made.

The address, data, and control lines can also be used to take care of special requirements of your digital assembly. For example, the address lines could be decoded and used with a write control line to latch the data you send out to the UUT. This could be useful for setting up input buffers on a card edge connector.

A Cable Tester

The following example demonstrates how non-μP digital testing can be implemented (see figure 4). The UUT is a cable with up to 64 lines. This particular UUT could be tested using most any interface pod. The Z80 pod was chosen only because it was convenient. On the left of figure 4 you will recognize the Z80 Pod Support Fixture circuitry of figure 1.

There are two 64-pin cable connectors shown; the lower one is the output (WRITE) side and the upper one is the input (READ) side. The data lines are buffered through line drivers (74LS541) to both the output latches (74LS374) and the input line drivers (also 74LS541). Only four address lines are used. A15 is used as a fixture select and A0-A2 allow selection among the eight different 8-line test groups on the cable. (Additional address lines can be used to expand the Cable Tester to a much larger number of cable lines.)

To use the Cable Tester to test a cable, simply connect the cable between the two connectors and WRITE selected data patterns to the addresses 8000-8007 (hex), and then READ the data at the other end. With a cable, two failure conditions are possible: lines that are open and lines that are tied together. We could write a Troubleshooter program to check for these two error conditions, but let's analyze the problem a little closer first.

The Cable Tester actually works as a RAM device: it latches (writes) data at a specified address and then tries to read it back. The Troubleshooter has powerful RAM-testing capabilities, so why not use them? For a simple straight-through cable, the Troubleshooter's built-in RAM LONG test will check for both of the failure modes, opens and shorts.

If the cable is expected to have interconnections among the lines, then a program can be written to check that those interconnections (and no others) are present.

This example is provided to help get you started on your unique applications. Your imagination's the only limit.

Detecting address - aliasing

by Tom Locke

The program to be described here is for users who have one of the new interface pods with the Quick RAM Test (i.e., 68000, 8086, or 8088), or for users who want to download an assembly-language RAM test into the Unit Under Test (UUT) RAM to test the RAM area faster. Although the Quick RAM Test will detect address-aliasing errors, it will not identify *which* address line caused the problem. This program will identify the faulty address line, and do it much more quickly than RAM SHORT. Users who write their own RAM test code should run this program *before* downloading that code into UUT RAM. This program will detect address-aliasing problems which could corrupt the downloaded code.

This program can also be used by anyone who just wants a fast address-aliasing test. The program as written prompts the user for the starting address, the ending address, the test address, and the address increment/toggle pointer. These values can be programmed in for frequent use on a common system. The test address is the failure address reported by the Quick RAM Test. If you don't have a suspect address, use the starting address for the test address. The address increment should be either 1 or 2 (0001 or 0010 binary). The address increment is used as the initial value of a pointer to the address bit to be toggled.

After inputting the four parameters, the program then zeroes the byte or word at the test address. The main program loop starts at Label 3. The first three lines do an exclusive-OR of the values of the test address and the toggle pointer. This effectively toggles the address bit indicated by the pointer.

If this new address is within the specified address range, a value of FFFF (hex) is written to the new address and the test address is read. If its value is still zero, the pointer is shifted left one bit and the test is repeated. If its value is *not* zero, the pointer is shifted left one bit and the
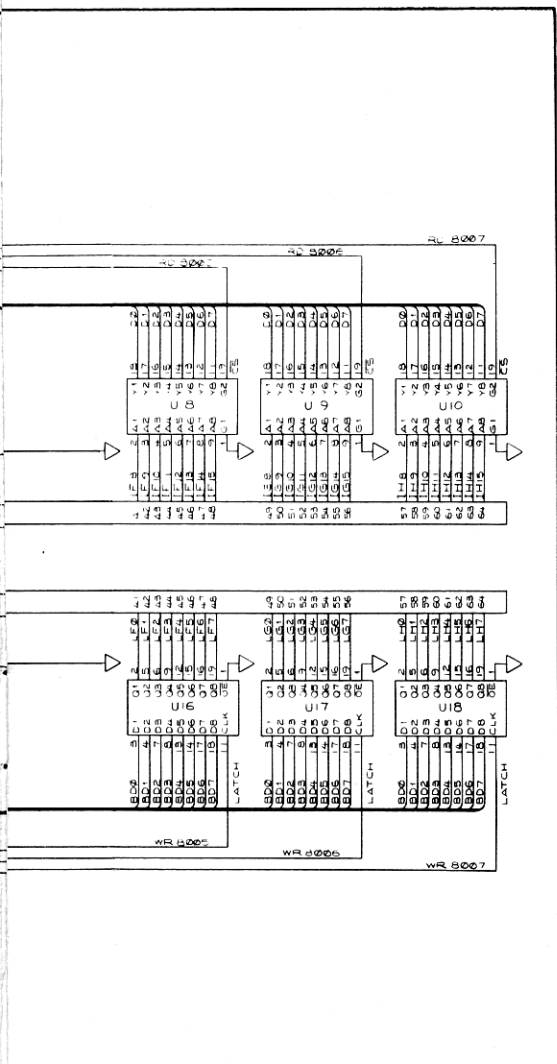
test is repeated. If its value is *not* zero, the address bit indicated by the pointer is flagged as the bad address bit.

If the new address is *less* than the starting address, the write and read are skipped, the pointer is shifted, and the test continues. If the new address is *greater* than the ending address. the program is terminated with no errors detected.

The listed program can easily be modified to read the special addresses associated with the Quick RAM Test to find its starting address, ending address, and error address. If a separate program executes the Quick RAM Test, it could call the new program when it receives an F1 error code (indicating an address-aliasing problem). This would provide a better diagnosis of which address line caused the problem.

STATEMENTS	COMMENTS
	! reg 8 = test addr
	! reg A = start addr
	! reg B = end addr
	! reg 1 = pointer to complemented bit
1: LABEL 1	
DPY-ENTER STARTING ADDRESS/A	! put start addr into reg A
DPY-ENTER ENDING ADDRES/B	! put end addr into reg B
IF REGB > REGA GOTO 2	! end addr > start addr, proceed
DPY-#ILLEGAL ADDRESS COMBINATION	
STOP	
GOTO 1	
2: LABEL 2	
DPY-ENTER TEST ADDRES/8	! put test addr into reg 8
DPY-ENTER ADDRESS INCREMENT/1	! addr inc:= start value of toggle ptr
DPY-LOOKING FOR ADDRESS ALIAS	! disp operating message
WRITE @ REG8 = 0	! zero test addr
3: LABEL 3	
REGF = REG1 CPL AND REG8	! next 3 lines do:
REG2 = REG8 CPL AND REG1	! reg F = reg 1 XOR reg 8,
REGF = REGF OR REG2	! which toggles addr bit selected by reg 1
IF REGF > REGB GOTO 6	! new addr > end addr, finished
IF REGA > REGF GOTO 4	! start addr > new addr, skip next write
WRITE @ REGF = FFFF	! write 16 1's to new addr
READ @ REG 8	! read at test addr
IF REGE > 0 GOTO 5	! test data > 0, disp error message
4: LABEL 4	
SHL REG1	! shift pointer to next bit
GOTO 3	! continue test
5: LABEL 5	
DPY-#ERROR AT ADDRESS BIT $1	
GOTO F	
6: LABEL 6	
DPY-NO ERROR DETECTED	
F: LABEL F	

Adapt your pod to different µPs

by Bob Cuckler

Microprocessor (µP) systems that are directly supported by the available Fluke 9000-Series interface pods are not the only systems that can be tested with 9000-Series mainframes. Many µPs have very similar pin-outs and operating characteristics. As a result of this, some µP systems can be interfaced to a 9000-Series mainframe through an interface pod meant for a different type of µP. This can often be done with a minimal amount of wiring or hardware (external to the interface pod). This hardware is called an interface pod adapter.

6500 Family

An Interface pod adapter can consist of something as simple as two DIP sockets connected together in a way that adapts the pin configuration of a particular interface pod to that of the µP in the system to be tested. Processors that are part of a generic "family" of µPs are often very good candidates for this kind of adapter. The 65XX family of µPs is a good example of such a processor family. The timing characteristics of the µPs in this family are the same; only the functions of each processor differ.

Several members of the 65XX family can be adapted from the 9000A-6502 Interface Pod using a pinout translation chart as a guide for constructing the adapter. You can construct adapters for 6503-, 6507-, or 6512-based systems using the pinout translation chart in Table 1 as a guide. These three processors were chosen as examples, but other members of the 65XX family can also be adapted to the 6502 Interface Pod using a similar chart.

For the 6503 and the 6507 adapters, several of the pins on the 6502 Interface Pod are not used. For example, the 6502 Pod has 16 address lines (AB0-AB15) and a valid address range of 0-FFFF (hex). The 6503 has only 12 address lines (AB0-AB11) and has a valid address range of 0-FFF. The 6507 uses 13 address lines (AB0-AB12) and has a valid address range of 0-1FFF. Both the 6503 and the 6507 lack the SYNC and S0 lines of the 6502. The 6503 and the 6507 also have only one clock output (as compared to the two-phase clock output of the 6502).

In addition, the 6507 also lacks the IRQ interrupt line.

The 6502 and the 6512 µPs are almost identical. The 6502 requires a single phase clock input and produces a two-phase clock output, while the 6512 requires a two-phase clock input and produces no clock output. The DBE output of the 6512 is not present on the 6502 (or the 6502 Interface Pod). Thus the adapted pod cannot support DBE activities of the 6512 Unit Under Test (UUT). On most 6512 UUTs, the DBE line is connected to CLK2; this arrangement will present no problem to 6512 adapter users. However, if the UUT relies on any DBE activity beyond the use of the CLK line, testing problems may result.

Signal Name	6502 Pod Pin	6503 µP Pin	6507 µP Pin	6512 µP Pin
Vss	1	2	2	1
RDY	2	5	3	2
CLK1(out)	3	—	—	3*
IRQ	4	3	4	4
■	5	—	—	5*
NMI	6	4	4	6
SYNC	7	—	—	7
Vcc	8	5	4	8
AB0	9	6	5	9
AB1	10	7	6	10
AB2	11	8	7	11
AB3	12	9	8	12
AB4	13	10	9	13
AB5	14	11	10	14
AB6	15	12	11	15
AB7	16	13	12	16
AB8	17	14	13	17
AB9	18	15	14	18
AB10	19	16	15	19
AB11	20	17	16	20
Vss	21	2	2	21
AB12	22	—	17	22
AB13	23	—	—	23
AB14	24	—	—	24
AB15	25	—	—	25
DB7	26	18	18	26
DB6	27	19	19	27
DB5	28	20	20	28
DB4	29	21	21	29
DB3	30	22	22	30
DB2	31	23	23	31
DB1	32	24	24	32
DB0	33	25	25	33
R/W	34	26	26	34
■	35	—	—	35*
■	36	—	—	36*
CLK0(in)**	37	27	27	37
S0	38	—	—	38
CLK2(out)	39	28	28	39
RES	40	1	1	40

Notes: A dash (—) means no pin exists
■ *6502 pin not named .*
 * *Do not connect pins 3,5,35, or 36 of 6512 µP to 6502*
** *Called CLK2 on 6512*

Table 1: 6502 Pod Adapter wiring guide

NSC800

Although Fluke does not manufacture an NSC800 Interface Pod, 9000-Series products can still be used to test and troubleshoot NSC800-based micro-systems. This example shows how an existing interface pod can be adapted (through the use of circuitry external to the pod) to an NSC800-based system. Even if you don't have to repair NSC800-based systems, this example might give you some general ideas about adapting existing 9000-Series Interface Pods to your particular micro-system.

The NSC800 features a bus structure and a signal set that are almost identical to those of the 8085 microprocessor (μP). The read/write cycle timing characteristics of the NSC800 are also very similar to those of the 8085. These similarities in bus structure and cycle timing make the 9000A-8085 Interface Pod the logical choice for adapting to an NSC800-based Unit Under Test (UUT).

The circuitry required for an adapter will depend on the requirements of the particular UUT involved. For example, if the NSC800 UUT does not use a certain signal line, the circuitry used to adapt that line to the 8085 pod will not be necessary. As you read through the example, you should refer to the timing diagram of Figure 1. Refer also to the schematic diagrams of Figures 2, 3, and 4.

NOTE: The NSC800 is a CMOS μP, which allows it to be operated at power supply voltages other than +5V. The 8085 pod (and the troubleshooter probe) operate at TTL signal levels. If the NSC800 UUT uses a power supply voltage other than +5V, level-shifting circuitry will have to be included for every line (except ground) connected between the UUT and the 8085 pod. This circuitry is not shown in the schematic diagrams of this example. If buffer circuitry is necessary, the ability of the pod to check for drivability errors may be affected.

The Clock Circuitry. The clock requirements for the NSC800 and the 8085 pod are basically the same: both μPs can be driven directly by a crystal or by a UUT-generated clock signal. The minimum and maximum operating frequencies of the NSC800 fall within those of the 8085 pod, so no clock division or multiplication circuitry is required.

Both μPs generate a single-phase clock output signal. A quick comparison of the clock signals for the pod (shown in Figure 1) and the NSC800 shows that the 8085 pod CLK output signal must be inverted for use on the NSC800 UUT. This will allow the 8085 pod bus signals to be synchronized with those on the UUT. If the CLK signal is not used on the UUT, no inverter is necessary.

NOTE: If a crystal is used on the NSC800 UUT for generating clock signals, the length of the 8085 pod cable lines (and the adapter cable lines) may make it necessary to include a clock oscillator in the adapter circuitry. For more information regarding the generation of 8085 Interface Pod clock signals, refer to the Fluke Technical Data bulletin titled "Guide to 8085 Microprocessor-Based System Testing." Contact your local Fluke Sales Office or Representative for a copy. Ask for Application Information bulletin B0151.
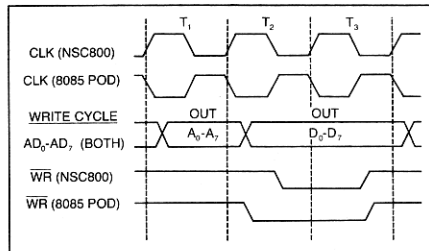

Figure 1. Timing differences between NSC800 and 8085 signals.

The Address and Data Buses. The address and data bus structures of the 8085 pod and the NSC800 are basically identical. Both multiplex the low-order address bits with the data bits. The timing of the multiplexing with relation to the clock signal is the same. The Address Latch Enable (ALE) signals are also identical. Therefore, no adapter circuitry is required for the address/data buses or the ALE signal.

Read ($\overline{RD}$) and Write ($\overline{WR}$) Control Signals. The $\overline{RD}$ control signal of the NSC800 is nearly identical to that of the 8085 pod. However, Figure 1 shows that the $\overline{WR}$ control signals *are* different. The $\overline{WR}$ pulse of the NSC800 begins one-half clock period (T-state) *later* than the $\overline{WR}$ pulse of the 8085 pod. The trailing edge of each write pulse occurs at the same point in the cycle.

The leading edge of the NSC800 $\overline{WR}$ pulse occurs when the data on the bus is guaranteed to be valid. This is *not* true of the 8085 pod $\overline{WR}$ pulse. If the NSC800 UUT only latches data on the trailing edge of the $\overline{WR}$ pulse, the above mentioned timing difference is not important. In that case simply connect the $\overline{WR}$ pin of the 8085 pod to the $\overline{WR}$ line of the UUT. However, if the UUT performs some special action on the leading edge of the $\overline{WR}$ pulse, the 8085 pod $\overline{WR}$ pulse may have to be modified such that the leading edge occurs when valid data is on the bus.

The $\overline{WR}$ pulse of the 8085 pod can be adjusted to begin at almost the same point as the $\overline{WR}$ pulse of the NSC800. Note from the NSC800 timing diagram that the NSC800 $\overline{WR}$ pulse has a duration of about one T-state. Thus a rising-edge-triggered D flip-flop can be used to delay the start of the 8085 pod $\overline{WR}$ pulse by one-half T-state. The CLK output of the 8085 pod is used to clock the flip-flop. The circuit necessary for this function is shown in Figure 2.
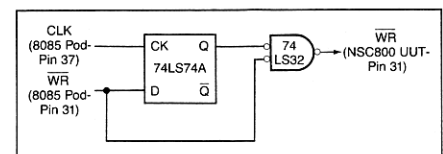

Figure 2. Circuit for adapting the 8085 Pod $\overline{WR}$ Signal to an NSC800 UUT.

Note that connecting the flip-flop between the 8085 pod and the NSC800 UUT defeats the pod's ability to check for drivability errors on the $\overline{WR}$ control line. For this reason, the circuitry of Figure 2 should be used only if the UUT requires it.

I/O Reads and Writes. The 8085 pod and the NSC800 feature identical input/output (I/O) bus structures. However, there is a difference in the timing of I/O operations. The NSC800 automatically extends the I/O read or write cycle by inserting a wait state. The 8085 does not have this characteristic; wait states must be generated by external circuitry. If the NSC800 UUT requires this extra wait state, the adapter will need to include the circuitry necessary for generation of an 8085 wait state. Figure 3 shows an example of a wait state generation circuit for the 8085 pod. The wait state will only be generated during an I/O operation. Note that the circuit of Figure 3 also allows wait states to be generated by the UUT through the use of the NSC800 $\overline{WAIT}$ line.
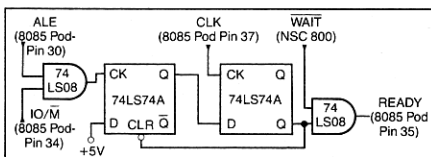

Figure 3. Wait state generation circuit for an 8085 Pod/NSC800 adapter.

If the UUT does not require the extended I/O cycle of the NSC800, simply connect the $\overline{WAIT}$ line of the NSC800 directly to the READY pin of the 8085 pod. (Refer to Figure 4.)
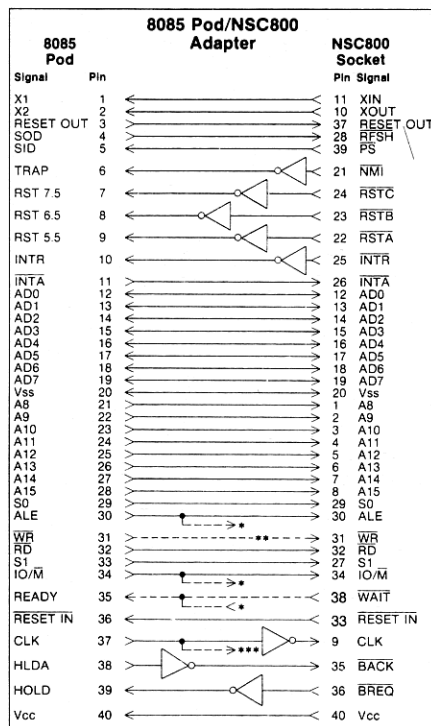

Figure 4. Adapter Wiring.

*See text and figure 3
**See text and figure 2
***See text and figure 2 and 3

Dynamic RAM Refresh Signals. The NSC800 μP features on-chip dynamic-RAM refresh control circuitry, but the 8085 pod does not support dynamic-RAM refresh. Few NSC800 UUTs use this capability, but if yours does, you might have to design refresh control circuitry into the adapter. In that case, the 8202 Dynamic RAM Controller should be considered for use in the adapter, as it is designed to be compatible with 8085 μP characteristics.

The Refresh ($\overline{RFSH}$) control signal of the NSC800 is used to alert circuits on the UUT that a refresh operation is taking place. If your NSC800 UUT uses the $\overline{RFSH}$ line, you may wish to use the 8085 pod to activate this line for testing purposes. To do this, connect the Serial Output Data (SOD) pin on the 8085 pod to the $\overline{RFSH}$ line on the UUT.

Connecting the $\overline{RFSH}$ line to the SOD line is useful only for testing the drivability of the $\overline{RFSH}$ line, using the WRITE CTL and BUS TEST functions of the troubleshooter. During normal troubleshooter operation, however, the $\overline{RFSH}$ line should be tied to Vcc, if it is to be held active.

The Power Save Line (NSC800). The Power Save ($\overline{PS}$) line of the NSC800 has no counterpart on the 8085 pod. However, many NSC800-based UUTs activate this pin to force the NSC800 into the "Power Save" mode. To use the 8085 pod to detect an active level on this line, connect $\overline{PS}$ line on the UUT to the SID (Serial Input Data) pin on the 8085 pod. A READ STS command from the troubleshooter keyboard can then be used to detect activity on this line.

Adapting Interrupt Lines. All of the interrupt lines on the NSC800 UUT can easily be adapted to the 8085 pod. Figure 4 shows that only inverters are used to adapt these lines. If the UUT does not use an interrupt signal of the NSC800, tie the corresponding pin on the 8085 pod to its inactive (low) level.

Adapting DMA lines. There is a slight timing difference between the DMA signals of the NSC800 ($\overline{BACK}$ and $\overline{BREQ}$) and those of the 8085 (HLDA and HOLD), but this timing difference is not critical. These lines can be adapted through the use of inverters (see Figure 4). If your UUT doesn't use the $\overline{BREQ}$ and $\overline{BACK}$ lines, tie the HOLD line of the 8085 pod to its inactive (low) level.

Constructing the Adapter. To construct the adapter, you'll want to use a Fluke Pod Adapter Packaging Kit. The kit is described in the "Now Available" section of this newsletter.

Using the Adapter with the 8085 Pod and the Troubleshooter. This adapter allows the use of virtually all of the functions of the 9000-Series Troubleshooters. Refer to the Operator Manual for instructions regarding the troubleshooter. Refer to the 8085 Interface Pod Instruction Manual for instructions about using the pod. Some points of caution are described in the paragraphs below.

The RUN UUT Mode. The RUN UUT mode of the troubleshooter cannot be used in the usual way. This is because the instruction set of the NSC800 μP is a superset of the instruction set of the 8085. Therefore the 8085 pod cannot execute NSC800 code.

In many cases, however, the RUN UUT mode of the troubleshooter can be used. 8085 instructions can be loaded into UUT RAM using the troubleshooter WRITE function. The RUN UUT mode can then be used to execute the 8085 code, starting with the initial RAM address of the code.

Status and Control Lines. The $\overline{RFSH}$, $\overline{INTA}$, RESET OUT, and $\overline{BACK}$ lines of the NSC800 UUT can all be "written to" using the WRITE CTL function of the troubleshooter. The $\overline{RSTC}$, $\overline{RSTB}$, $\overline{RSTA}$, $\overline{PS}$, $\overline{RESET}$ $\overline{IN}$, $\overline{INTR}$, NMI BREQ, and WAIT lines of the UUT can all be "read" using the READ STS function.

Remember that those NSC800 μP input or output lines with inverters between the UUT and the pod will have *opposite* Status or Control bit values.

The Probe. The troubleshooter probe can be used normally on the NSC800 UUT — *if* the UUT signal levels are compatible. CMOS signal levels are probe-compatible if the CMOS circuitry is operated with Vcc equal to +5 volts.

2650 UUTs

This article describes an interface pod adapter that allows 9000-Series Micro-System Troubleshooters to test and trouble-shoot a popular slot machine controller. The controller is based on the Signetics 2650 microprocessor (μP). Since Fluke does not manufacture an interface pod for 2650-based systems, it was necessary to construct the adapter to allow an existing interface pod to communicate with the Unit Under Test (UUT).

The Z80 Interface Pod was chosen for use with this particular adapter due to some basic similarities between the Z80 pod and the 2650 μP. For example, the Z80 pod has a 16-bit address bus, and the 2650 has a 15-bit address bus. Both the Z80 pod and the 2650 μP have a separate 8-bit data bus. Both require a single-phase clock signal. Also, the timing

of a Z80 pod bus cycle is similar to the timing of a 2650 bus cycle. These similarities allowed many Z80 pod lines to be connected directly to the UUT, including the address and data bus lines. Thus output lines connected directly between the pod and the UUT can be properly checked by the pod for drivability errors.

NOTE: *The 9000-Series Trouble-shooters include an automatic "drivability" diagnostic in most tests. Direct connection of the μP output lines to the interface pod allows the Troubleshooter automatic tests to check the UUT for the proper "drivable" condition of these output lines. Having to include a buffer or other logic on these lines would mean that the automatic bus-drivability tests would check the buffer circuitry on the adapter rather than the UUT circuitry, and actual UUT errors of this type would then be reported as errors in the RAM, ROM, or I/O tests.*

Figure 1. Z80 POD/2650 Adapter Circuit Diagram.
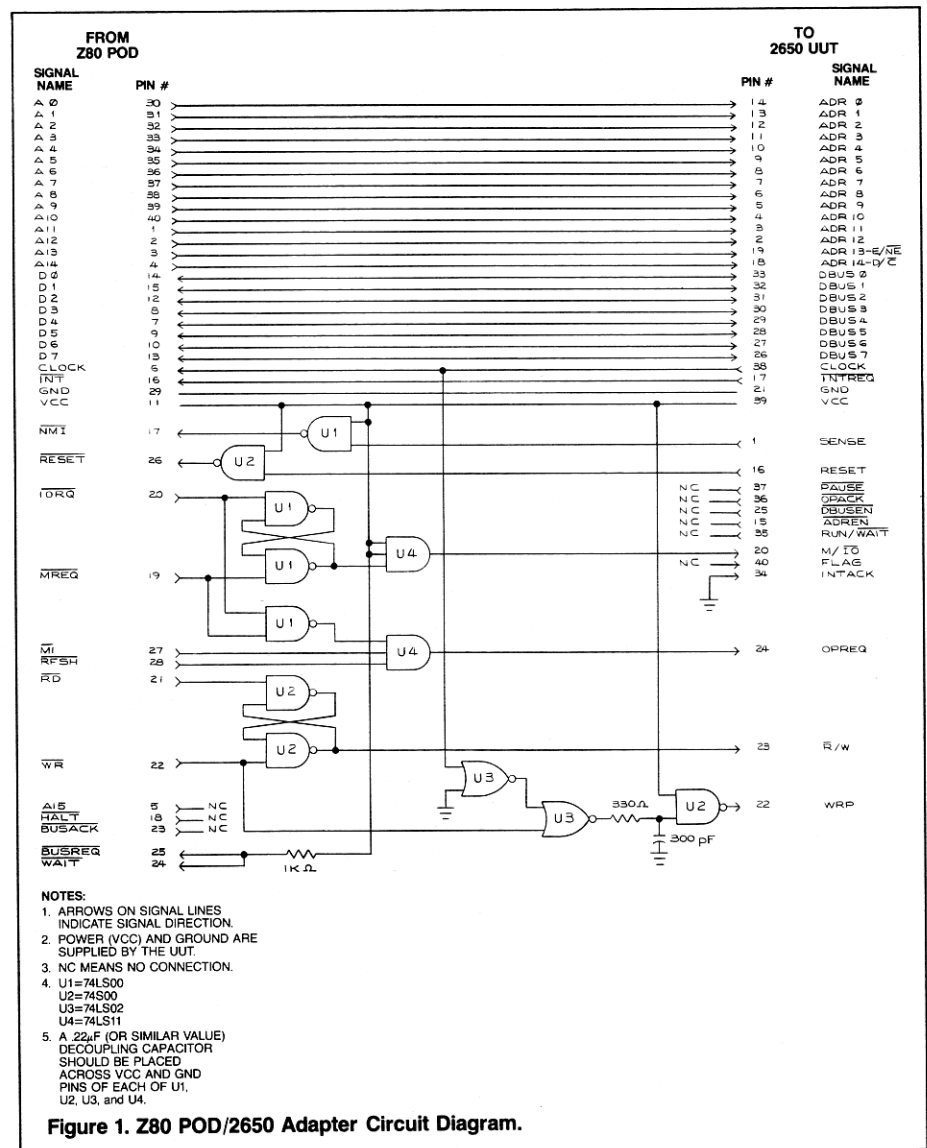
The principle design goal of any interface pod adapter should be to allow the pod to perform individual read and write cycles on the UUT. To achieve this goal with the slot machine controller, several Z80 pod control lines had to be adapted with digital logic to emulate the control lines of the 2650 μP. A schematic diagram of the adapter circuitry is shown in Figure 1. Note that the adapter circuitry is really rather simple, considering the differences between the Z80 pod and the 2650 μP.

NOTE: *In 9000-Series literature, control lines are lines (other than address or data lines) that are output from the pod. Status lines are lines (other than data lines) that are input to the pod.*

To allow read and write strobes, the $\overline{WR}$ and $\overline{RD}$ control signals of the Z80 pod are fed into a SET/RESET flip-flop constructed from two NAND gates. The output of this flip-flop emulates the $\overline{R}/W$ line of the 2650. The same scheme is used to create the M/$\overline{IO}$ line of the 2650 from the $\overline{MREQ}$ and $\overline{IOREQ}$ lines of the Z80 pod. The timing characteristics of these arrangements are slightly different from those of the 2650 μP, but this is often not critical and has worked well on this slot machine controller.

The OPREQ line of the 2650 is emulated by the logical combination of several Z80 pod control lines. The WRP line is emulated by the logical combination of the CLOCK and $\overline{WR}$ lines of the Z80 pod. Note that the RC circuit between the two gates (that create the WRP line) is used to create a small amount of delay. This allows the timing characteristics of the WRP line to be emulated more accurately.

To allow the troubleshooter mainframe to report the activity of the SENSE, RESET, and $\overline{INTREQ}$ lines, each of these three lines is connected to the appropriate status line input of the Z80 pod. The RESET and SENSE lines are inverted between the UUT and the pod to match the logic level convention of the corresponding Z80 status line inputs. Refer to the Z80 Interface Pod Instruction Manual for information on the methods used by the pod and the Troubleshooter to report activity on these lines.

Two Z80 pod address lines are used to emulate two of the 2650 I/O control lines. The A14 line of the Z80 pod emulates the ADR14-D/$\overline{C}$ line of the 2650. The A13 line of the pod emulates the ADR13-E/$\overline{NE}$ line. This results in the address space of the Z80 pod being partitioned into additional sections as defined in Table 1 below.

ADDRESS	DESCRIPTION
0 -7FFF	Memory Address Space
10000	I/O Control
12000	I/O Data
14000 - 140FF	I/O Extended (140xx, where xx is the desired 8-bit 2650 I/O address)

Table 1. Z80 Pod/2650 Adapter Address Spaces.

The complete address space of the Z80 pod consists of 0-FFFF for memory accesses and 10000-1FFFF for I/O accesses. Note that this address space is not fully used by the 2650 adapter. Attempted accesses outside the valid adapter address space may not work — even if the access is still within the valid Z80 pod address space.

Note that seven 2650 μP lines (IN-TACK, $\overline{PAUSE}$, $\overline{OPACK}$, $\overline{DBUSEN}$, $\overline{ADREN}$, RUN/$\overline{WAIT}$, and FLAG) aren't emulated by the adapter. These lines aren't used on this UUT (or don't need to be adapted for test purposes), and therefore don't need to be adapted to the pod. Also, the Z80 pod lines $\overline{HALT}$, $\overline{BUSACK}$, and A15 are not connected to the adapter, and the pod lines $\overline{BUSREQ}$ and $\overline{WAIT}$ are tied to Vcc. None of these lines are required by this particular UUT.

This adapter successfully performs reads and writes on the UUT for which it is designed. Remember that most other 9000-Series functions such as ROM TEST, RAM TEST, RAMP, WALK, etc. are composed of individual reads and writes on the UUT; those functions have been used with equal success. The 9000-Series RUN UUT function cannot be used properly, however, since the instruction sets of the 2650 and Z80 μPs are not compatible.

This adapter was designed for use in one particular 2650 application, but the ideas presented in this article might be useful for other 2650 users — and other pod adapter designers as well. If you are interested in constructing your own pod adapter for a 2650-based micro-system, you should carefully determine what your UUT's testing and timing requirements are

versus the characteristics of the adapter described in this article. This adapter may not work "as is" on your 2650-based UUT, and even a different interface pod may be a better choice for your adapter. For more information, consult the 2650 μP Data Book and the appropriate Fluke interface pod manuals. Also, a Technical Data Bulletin is available on Interface Pod Adapters. Contact your nearest Fluke Sales Office or Representative and ask for Technical Data Bulletin #B0156.

Remember that Fluke provides the 9000A-200 Pod Adapter Packaging Kit to make the building of a pod adapter easier. The kit contains the necessary hardware for mounting and housing adapter components and for connecting the adapter between the Interface Pod and the UUT. Contact your nearest Fluke Sales Office or Representative for pricing and delivery information.

LEARN:
A detailed discussion

by Mike Renneberg

The LEARN algorithm is a tool designed to be used when first encountering a new Unit Under Test (UUT). It provides the information that is needed when using the automatic RAM, ROM, and I/O (input/output) tests and can provide hints to the technician on how the microprocessor (μP) system is structured (to supplement typical UUT documentation). The LEARNed information can also be entered manually or changed via the VIEW keys on the 9010A keyboard. Although the LEARNed information is very simple, the LEARN algorithm is not. The complexity is due to the fact that there are probably as many different μP system designs as there are designers and each one uses different tricks to enhance performance or to reduce cost.

The LEARN algorithm automatically determines the locations of RAM, ROM, and I/O by performing write and read operations at *all* possible memory addresses. At each location, it stimulates the UUT and determines which bits, if any, are read-writable. Memory is tested in 64-byte blocks, and those that are identifiable are classified as RAM, ROM, or I/O, using the following rules:

RAM Blocks of memory that are at least 64 continuous bytes long in which all bits are read-writable and all locations are distinct (i.e., writing to any location will not affect the contents of any other location of the 64 bytes).

ROM No bits are read-writable, and there appear to be 64 bytes that are fully decoded (i.e., all 64 locations appear to be distinct from any other block of 64 bytes).

I/O One or more bits are unconditionally read-writable, but one or more of the conditions for RAM are not met. The bits that are read-writable are the reported I/O bits.

The 9010A LEARN algorithm analyzes the data read and eliminates certain types of data that fail the above three tests. Addresses containing *only* the following types of data are thus eliminated and *not* reported by LEARN:

1. All bits of each address are logic zero, such as that data read from unused addresses when pull-down devices are connected to the data bus.
2. All bits of each address are logic one, such as that data read from unused addresses when pull-up devices are connected to the data bus.
3. The data byte at each address exactly matches the low-order bits of the address word, such as that found under certain circumstances upon reading from μPs with a multiplexed address and data bus (e.g. 8085).
4. All address blocks of 64 bytes exactly match another, previously recorded, block of 64 bytes. If RAM, writing to any address in one block will also write the same data to a duplicate address position in the previously located block. If ROM, all data read from one block exactly matches the data in a previously located block. (This is very commonly found in μP systems where only a portion of the address space available is used and the product designer has not decoded all bits of the address word — causing a mirror image of the RAM or ROM to appear in address space that is not used).

The multiplexed bus

In order to improve package size and density some of the newer μPs have implemented a multiplexed bus configuration where some of the address bits are multiplexed with the data bits on the same pins of the μP (e.g., 8085 and 8086 μPs). In a typical READ operation the μP first places the low order bits of the address word on the bus, for external circuitry to latch, and then enables the external circuitry to drive the bus with the data to be read.
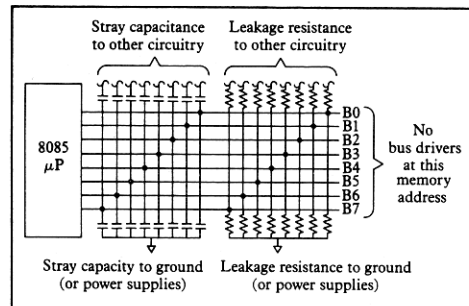

Fig. 1

For those address locations that are not used in a particular UUT, nothing drives the data bus, and it appears as in Figure 1 (an open or tri-state bus condition). During a read operation to one of these unused addresses, the μP first places the low order bits of the address word on this bus, then attempts to read a data byte from the bus. Under most circumstances the data read from this "floating" bus will be one of the easily eliminated types discussed above. However, it is difficult to control the stray capacitance and leakage resistance of the bus, and, depending on these parasitic values, the data can appear as something other than the trivial data. Bus lines with low-value pull-up or pull-down devices quickly change to the pulled level after the address byte has been removed from the bus and before the μP performs the remainder of the read operation. Bus lines with high impedance pull-up or pull-down devices (or none at all) change state slowly, and may still retain the low byte of the address at the instant the μP is completing the read operation. In either of these cases the LEARN algorithm will recognize this condition and eliminate these addresses from the reported memory map.

However, because these parasitic parameters are difficult to control (dust accumulation or a wet fingerprint can represent a significant amount of leakage resistance), it is possible for one or more of the data lines to float in a logic direction different than the others. This causes the read operation to obtain data that appears as acceptable data and the LEARN algorithm to report a block of ROM data — since the data appears as ROM data would. The data read is very dependent on the sequence and timing of the read operations. The sequence of operations in ROM test is different than in LEARN, and thus the non-existent ROM addresses will usually fail the ROM test, even when done on a known-good UUT. This ROM data is usually reported as many small ROM blocks and thus is easily recognizable. They are also easily eliminated by using the VIEW ROM and DELETE keys on the 9010A. This characteristic only appears in the LEARN algorithm because LEARN, by its very nature, must access all possible μP address space, while the UUT or the 9010A, under normal operation, never accesses the unused address space.

If, when using LEARN, many small blocks of ROM are reported and it is suspected that this is the cause, then providing low-value pull-up or pull-down devices on the data bus will easily eliminate the false ROM reporting. (Resistors with a value between 1 kilohm and 20 kilohms are usually adequate for this purpose.) This can easily be implemented with a socket adapter that plugs into the μP socket. Or you can use a clip-on device that attaches to another device on the μP data bus. All multiplexed data lines and either a +5-volt line or ground must be available. If the system design includes bus buffers, the pull-up or pull-down devices are attached to the same side of the buffers that other bus drivers are, i.e., the same side as the RAM and ROM components (Figure 2). These pull-up or pull-down resistors are only attached when using LEARN and are not needed for any other testing.
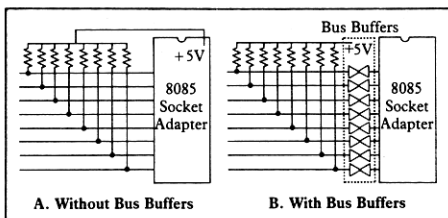

A. Without Bus Buffers   B. With Bus Buffers

Fig. 2

Finding the elusive write-only-memory (LEARN and the system I/O)

I/O circuitry is very much custom-designed for each product application. As discussed before, LEARN is designed to locate I/O that is read-writable at a particular address, whether one bit or all bits. If the μP implements special I/O memory, separate from RAM memory space (as with the 8080, Z80 and 8085), the 9010A will exercise this also and report the read-writable I/O it finds. Many common PIA's contain read-writable registers and thus are reliably reported by LEARN. This information, used with a schematic, gives hints as to how the I/O space may be organized. The limitations of LEARN are primarily due to the fact that closing the loop between stimulus and response can occur only via the μP socket. Thus write-only and read-only I/O registers are not detected.

As mentioned before, there are as many techniques of organizing UUT address spaces as there are μP system designers and, therefore, μP systems can be very complex. LEARN is one of the many tools within the 9010A that help the technician understand a new product being tested for the first time.

RAM SHORT or RAM LONG... which, when, and

To meet the needs of a wide variety of users, the 9010A contains two Random Access Memory (RAM) testing programs. RAM SHORT provides a fast test for finding shorted or open bus lines and gross failure of RAM chips. RAM LONG provides a complete checkout of all RAM circuits and components.

The RAM tests on the 9010A do not need specific knowledge about what type of RAM is in the Unit Under Test (UUT). The tests work equally well with static or dynamic RAMs from all manufacturers. Since RAM accesses occur at the speed of the microprocessor in the UUT, the dynamic characteristics of the RAM are exercised, revealing faults that might otherwise go unnoticed.

RAM SHORT performs these tests: (1) a read/write test (to check the basic functioning of the RAM), (2) a tied-data-lines test (to locate shorted or open data lines), and (3) a fast decoder test (to locate shorted or open address lines).

RAM LONG provides (1) a read/write test, (2) a tied data lines test, (3) a thorough decoder test, and (4) a pattern sensitivity test (to find subtle, intermittent RAM cell failures.

The tests are performed in order. If any test fails no further tests are performed, because subsequent test results cannot be trusted if previous errors have not been corrected.

The read/write test is performed by both RAM SHORT and RAM LONG. This test insures that each bit at every RAM address can store both 1 and 0. At each address in the tested range, a zero is written to the RAM, and then the location is read to check that all bits are zeroes. A word of all 1's is then written to the location and read back to check that all bits are 1's. If there are any bits that cannot be driven to both states, a diagnostic message is printed.

The tied-data-lines test also is performed by both RAM tests. This test checks that data lines to the RAM chips are not tied together. The test is performed at the *first* address in the tested range. Each data line is tested by being written to the

why?

by Kurt Guntheroth

opposite state from all other data lines. Then the data is read back. For example, on an 8-bit data bus, the following patterns would be written:

$$\left.\begin{array}{c} 00000001 \\ 11111110 \end{array}\right\} \text{data bit 0 tested}$$

$$\left.\begin{array}{c} 00000010 \\ 11111101 \end{array}\right\} \text{data bit 1 tested}$$

- -   -
- -   -
- -   -

$$\left.\begin{array}{c} 01000000 \\ 10111111 \end{array}\right\} \text{data bit 6 tested}$$

The last data bit (7) does not need to be tested since it would already have been reported as tied to some other bit. If a data line is tied to the tested line, one of the lines will be pulled into a state opposite to the one in which it was written. For example:

	Data Bit 3 Test	
Data Written	00001000	11110111
Data Read	00001000	10110111
		↑ ↑
		bits tied

In this example, data line 6 was forced low when data line 3 went low, indicating that data lines 3 and 6 are tied together.

RAM SHORT contains a fast decoder test. A pattern of data bits is written at each address in the range to be tested. This pattern contains information about the address at which it was written. Next, each location is read back to see if the data there is what was originally written. If the data read back matches the pattern that was written, everything is fine. If the data read back differs from the written pattern, it is because two locations in RAM have become connected or "aliased" to each other. The typical cause of this is failure of the address-decoder circuitry — an address line either shorted or open, or a failure of

internal RAM decoding logic. The faulty address line is identified in a diagnostic message. Its functioning can be quickly verified with the probe.

RAM LONG uses a different decoder test which tests each address location in the following way: The current location is first zeroed, and then a word of all 1's is written to each location which differs from the current location by a single address bit. After each of these locations is written, the current location is read to check that all data bits are still 0's. If any bit has changed to a 1, the current location is aliased to the test location in that bit. As above, address-aliasing is caused by an open or shorted address line or internal failure in the RAM chip.

RAM LONG also contains a test for pattern sensitivity. A RAM cell is pattern sensitive if writing a certain data pattern at one address causes a change in data at another address. Pattern sensitivity may be caused by manufacturing defects in the RAM chip, operation at temperature extremes, or marginal circuit design. Repair of a pattern-sensitivity failure generally involves replacing a RAM chip. In the pattern-sensitivity test, the RAM is written with a sequence of data patterns that guarantees to drive any two bits in the tested address range to opposite states at some time during the test. Thus, if writing to one cell causes a change in any other cell in the RAM, the pattern-sensitivity test will report this fact.

RAM SHORT detects all failures of external address-decoder circuitry and finds data lines that are stuck or tied together. It also finds the majority of RAM chip failures, because these failures typically have obvious symptoms. There are some failures that RAM SHORT is not able to detect. If RAM SHORT passes, and the technician still suspects a RAM failure, then RAM LONG should be run.

The length of time RAM SHORT takes to run is directly proportional to the number of words or bytes of memory tested. This means that running RAM SHORT on 16K bytes of RAM takes just 16 times as long as running RAM SHORT on 1K bytes of RAM. RAM LONG, on the other hand, takes time proportional to the number of words tested times the

number of patterns it writes, which increases with the size of the address range tested. This means that it takes 22 times as long to test 16K bytes of RAM as it does to test 1K bytes and over 100 times as long to test 64K bytes as it does to test 1K bytes.

It is very unlikely that a pattern-sensitivity failure will occur in two RAM locations that are on separate chips. This means that if you have a 32K-byte block of RAM that is composed of four 8K-byte banks, running RAM LONG on the four banks separately is faster and just as effective as running it on a single bank of 32K bytes.

Both RAM tests perform the tied-data-lines test only once, at the first address in the tested range. However, if the RAM being tested consists of several banks of RAM chips, tied data lines on some banks will not be diagnosed correctly. For example, consider a system with two banks of RAM, where all data lines are OK to the first bank of RAM, but one line is open to the second bank. A single RAM test over the entire address range will not correctly diagnose the stuck data bit, because the tied-data-lines test is performed only at the first address of the first bank of RAM. The problem will, instead, be incorrectly diagnosed as a read/write error in that data bit. A RAM test over the second bank alone would correctly diagnose the problem as a stuck (open) data line. If a read/write failure is reported at the beginning of a data bank, the test should be repeated for that bank alone.

RAM LONG may be the most economical use of your time if you have a board that is failing due to a suspected failure of RAM chips. RAM SHORT is more appropriate for troubleshooting completely dead boards or for production testing.

New µP/pods test large RAMs 20 times faster

by Tom Locke

The 68000, 8086, and 8088 interface pods have built-in Quick Memory Tests to allow the user to test the potentially large memory areas much faster. In addition to greatly reducing test time, the Quick Memory Tests also provide: (a) a choice of byte or word tests for the 16-bit microprocessors (µPs), (b) a choice of address increment size, (c) automatic checking for inactive data bits in the Quick ROM Test, and (d) more flexibility under program control. Each of these capabilities is discussed in detail below, followed by a short explanation of how to use the tests, and a comparison with the normal troubleshooter tests.

The speed was increased by putting more intelligence into the interface pod, eliminating the need for communication between the pod and the troubleshooter on every transaction. This reduces the RAM test time by a factor of about 20.

The combined choice of byte or word addressing and of address increment size gives the user more flexibility. For a RAM test, he can specify byte addressing with an address increment of 1, thus testing every byte individually and checking the ability of the UUT to do single-byte writes without altering the adjacent byte. This mode also causes more activity during the RAM test, increasing the chance of detecting pattern-sensitivity errors. For a ROM test, he can specify byte addressing and an address increment of 2, allowing separate checksums for individual ROMs on either the high or low byte of data. The address increment of 2 specifies every other byte, thereby testing either all odd or all even addresses, depending on the value of the starting address.

The Quick ROM Test will look to see that every data line is valid both high and low sometime during the test. If not, it will set an error flag and allow the user to see a mask of which line, or lines, were inactive. An inactive data line doesn't always indicate an error, but when an incorrect checksum is received, this flag and mask will be strong evidence of a shorted line or a bad data buffer.

The added flexibility under program control is a result of the way

these tests are implemented. Because the Quick Memory Tests are contained in the interface pod and not in the mainframe, they cannot be accessed directly by the normal troubleshooter test keys. Instead they are initiated, controlled, and examined by reads and writes to "special" addresses (outside the µP's address space). When the pod receives an instruction to read or write at one of these special addresses, it is interpreted as a special function command. Thus all commands and examinations are just a series of reads or writes as far as the troubleshooter is concerned. This allows full control and decision-making while under program control.

Following this paragraph is a short explanation of these tests. For more details, look in the Operating Information section of the appropriate interface pod instruction manual, under Special Functions. If you haven't read the Operating Information section yet, we recommend that you read the entire section. It contains other worthwhile information unique to operating your particular interface pod.

Two commands are required to initiate a Quick Memory Test: the first one to specify the starting address of the block to be tested, and the second one to specify the ending address and the address increment. The format of each command is as follows:

WRITE @ t aaaaaa = x y , where

t = test code (2 for Quick RAM Test, 3 for Quick ROM Test)

aaaaaa = beginning or ending 6 digit address

x = address increment (only when specifying the ending address. 0 means use default increment)

y = address type (0 for starting address, 1 for ending address)

The complete address is prefixed with a test code (t) to specify whether RAM or ROM is to be tested. The write data specifies the address type (starting or ending). When specifying the ending address, the higher order digit of the write data

specifies the address increment to be used. If the write data is 01 (or just 1), the default address increment specified in the pod manual is used. The pod initiates the test upon receiving the ending address, so the starting address must be specified first.

NOTE: With some microprocessors (e.g., 8086, 68000) the address specification determines the data size (word or byte). Refer to the appropriate Interface Pod Instruction Manual for details.

For example, to perform a Quick RAM Test on an 8086 pod over the address range of 0 to FFFF (hex) using *byte* addresses and an *address increment* of 1, execute the following two instructions:

WRITE @ 2800000 = 0
WRITE @ 280FFFF = 11

The first instruction specifies the starting address and the second specifies the ending address and the increment, and begins the test.

To see the progress or results of the test during or after execution, press READ, then ENTER. In general, Ax means test Aborted, Bx means Busy (test not finished), Cx means test Complete (no errors), and Fx means test Failed. These codes are explained in more detail in the manual. If the test failed, or you're just curious, related data is available at other special addresses as described in the manual. This data includes the specified starting and ending addresses and the status code. The ROM test also makes available the checksum of the last block tested and a hex mask of any inactive bits. The RAM Test data includes the failure address, the expected data at that address, the actual data read, and a hex mask of bad data bits.

The Quick RAM Test will find any errors that RAM SHORT will find. It will also allow testing the RAM space of a 16-bit µP one byte at a time, which RAM SHORT/LONG won't. It is a two-pass RAM test. The first pass (pass 0) tests the read/writability of every bit, both high and low. A status code of F0 means the test has failed the first pass and has found a RAM location which has a read/write

problem. The second pass (pass 1) reads every location again, looking for the unique data last written into that location. A failure of this pass usually means an address-aliasing problem, but could also be caused by noise, refresh, or pattern-sensitivity problems. This pass will find *all* hard address-aliasing problems, but it is not as good as RAM LONG for detecting or identifying soft errors (i.e., intermittent, pattern-sensitive, etc.).

The Quick ROM Test provides a checksum of the block tested. Though not as complete a test as the CRC signature of the troubleshooter, it is sufficient to detect most ROM errors.

Because of the extra address encoding and having to read at special addresses to see the results, the Quick Memory Tests are a little harder to use in Immediate Mode than the standard troubleshooter tests. But programs can be written to make execution of these tests as easy as executing standard tests. When large blocks are tested, the greatly enhanced speed will justify the small increase in operator difficulty.

9010A & 9020A. What's the difference?

by Ken Hallmen

That is a question we often hear about the 9020A and the 9010A. When 9010A and 9020A Micro-System Troubleshooters are sitting side by side, the most obvious difference is the keyboard. The 9020A doesn't have any of the Test Sequencing, Arithmetic, Tape, or AUX-IF keys of the 9010A. The LEDs on the right side of the display on the 9020A are also different, replacing the PROGMING and EXECUTING LEDs of the 9010A with REMOTE and SRQ on the 9020A. The 9020A also does not have a tape drive.

A major functional difference is that the 9020A has a machine-to-machine software interface, whereas the 9010A was designed to enable the user to obtain information directly about the 9010A memory and operator-entered programs. The 9010A may have an optional RS-232 interface which allows the user to print out a stored memory map, listings of programs, and setup parameters. It also allows the uploading or downloading of the entire stored program memory. The 9020A, on the other hand, can be configured with either an IEEE-488 or an RS-232 interface. It was designed to work in conjunction with a system controller, allowing the system controller to gain access to the Unit Under Test (UUT) for functional testing and troubleshooting.

Operation of the 9020A may be thought of as a sequence of remote keystrokes. All of the keyboard functions available in local operation are duplicated by remote commands, including complete control over the probe stimulus. In addition, there are functions available only through the remote interface, such as BLOCK READ and BLOCK WRITE. The controller provides the command sequencing and stores these commands in its own storage medium. Since all commands for the 9020A originate from the controller, the 9020A does not need to have memory for storing programs.

The keyboards on both the 9010A and the 9020A allow the user to interact with the UUT to troubleshoot faults. The error messages that appear on the display are the same for both models. When the 9010A detects a hardware error, such as a read/write error in a RAM test, it presents the operator with an error message. It is then up to the operator to take some action to troubleshoot the UUT. If the RAM

test was run from a program, program execution is suspended until the entire list of errors is presented to and handled by the operator. The 9020A, on the other hand, when it is being remotely controlled, sends error status information to the controller, allowing the controller to handle different types of errors differently. When the 9020A detects a read/write error, for example, it will send the first error status to the controller. The controller can then abort the RAM test and immediately start troubleshooting the UUT by guiding the operator in the use of the probe.

Can 9010A Programs Run on the 9020A?

The quick answer to this question is "no". The 9010A was designed as a completely integrated stand-alone hardware and software environment. The 9020A was designed to be part of an integrated test and troubleshooting system. The 9020A accepts ASCII read or write commands from the IEEE-488 bus or from an RS-232 port. It then performs the indicated operations on the UUT hardware and returns information to the controller. In the case of a READ command, the status is returned to the controller first, followed by the data read from the UUT.

One difference between the 9010A and the 9020A is that, in the EXECUTING mode, the 9010A performs the read or write operation on the UUT and stores the data read in Register E. Register E can then be tested by the 9010A program and the flow of control modified according to the results. The 9020A, on the other hand, only performs the read or write operation on the UUT and then sends results back to the controller.

What Can You Accomplish Using the 9020A?

Programming a test on the 9010A is very simple to do but the language for troubleshooting faults is somewhat limited. The 9010A has comparatively few registers for storing intermediate results of a test. Because of this, the 9010A does not have an easy way to store a large number of signatures or levels from a good UUT to

compare against an unknown UUT. The 9010A language controls the flow of testing in a very easy-to-understand method by using GOTOs and LABELs. But modern system controllers, like the Fluke 1720A, utilize a richer control structure, one which allows the testing and trouble-shooting of UUTs to be much easier. The 1720A also makes available hundreds of different variables and thousands of subscripted variables in the Enhanced Fluke BASIC language.

Because of the capability of the 9020A to handle any error in the UUT and still allow the controller to control the testing, it is possible to generate complete Guided Fault Isolation (GFI) test programs for the UUT. The controller can store signatures and levels from a known good UUT, allowing straightforward generation of 9020A GFI test programs.

Using the 9020A, hybrid boards with both analog and digital sections can be checked out at the same production test station, as shown in figure 1.

The controller can use a system counter to verify the clock rate of the microprocessor (μP) and of any programmable timer outputs from the UUT. First the BUS, ROM, and RAM are verified and then the I/O testing can begin. The analog outputs can be routed to a system digital multimeter, allowing gain or offset adjustments to be made. In most systems with analog inputs and outputs, these analog nodes can be statically controlled, because the related data is latched, making the timing associated with the analog nodes unimportant. In these cases, the 9020A can set up the output and the system DMM can measure the programmed value. In some situations, however, the μP must perform a series of reads and writes in a very precise manner to insure that the analog measurement or stimulus will be fully tested "at speed". In these cases, the necessary μP code (contained in the UUT ROM or downloaded into the UUT RAM) can be executed using the RUN UUT command.
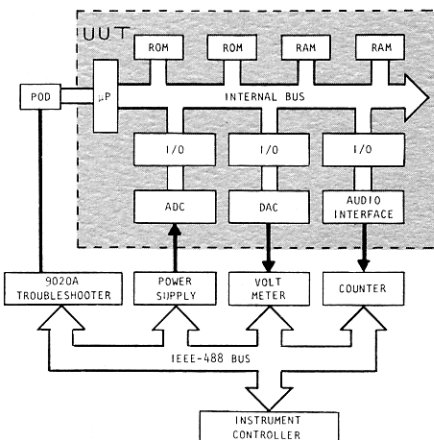

Figure 1. Hybrid Test System

Design Margin Verification

One of the more challenging parts of the design process for any μP-based system is the verification of the allowed margins of error, called design margins. These design margins include parameters like the allowable timing skew in signals arriving at a gate, or the effective remaining noise threshold at any logic node. Small design margins are often the cause of latent hardware problems in the field. By changing the conditions under which the hardware operates, such problems can be brought to the surface and fixed. If the hardware is tested as a system in its normal configuration (not as separate PCBs), the variation of components and their mutual interactions can be fully tested. Tests of these margins need to be made in more than just one location in the system. Besides the μP-associated circuitry, other logic and I/O sections require many separate tests.

A unique testing procedure for verifying system design margins over the IEEE-488 bus is possible using the Fluke 9020A. An IEEE-488-based automatic test system can control the system clock frequency, the supply voltages, and the ambient temperature (see figure 2). The first step would be to set the supply voltages and ambient temperature to nominal values and vary the clock frequency to check the timing margins. The 9020A can be used to verify that the ROM, RAM, and associated circuitry are all meeting the necessary timing requirements. Then, as the clock frequency is slowly increased, at some point the hardware will fail one of the ROM or RAM tests. This gives a good indication of the margin of error for the timing under one set of test conditions.

Now, by changing the supply voltages and repeating the same tests, the margin of error can be checked against supply voltage variations. The last parameter to be varied is the ambient temperature. Since temperature stabilization of the system can take a long time, test time can be reduced if the other parameters are varied at each ambient temperature setting. Examining many systems can increase the level of confidence in the results of these tests.
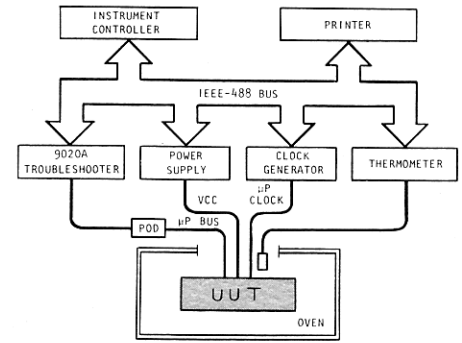

Figure 2. Design Margin Test Environment

Repetitive design-margin testing, if done manually, can be very time-consuming, tedious, and prone to data errors. The 9020A Micro-System Troubleshooter, combined with an IEEE-488 system to control the clock frequency, power supply voltages, and temperature, provides a simple means to check the design margins for these three test parameters. If other parameters are important to the operation of your system, then additional IEEE-488-controlled stimulus or measurement equipment will be required. The system can be subjected to continuous testing while the results are monitored over the duration of a life test. To increase the number of UUTs tested, multiple test systems on a common IEEE-488 bus can be employed and controlled by a single controller. System configurations employing the 9020A provide for faster throughput, error-free data gathering, error data manipulation, and hardcopy results.