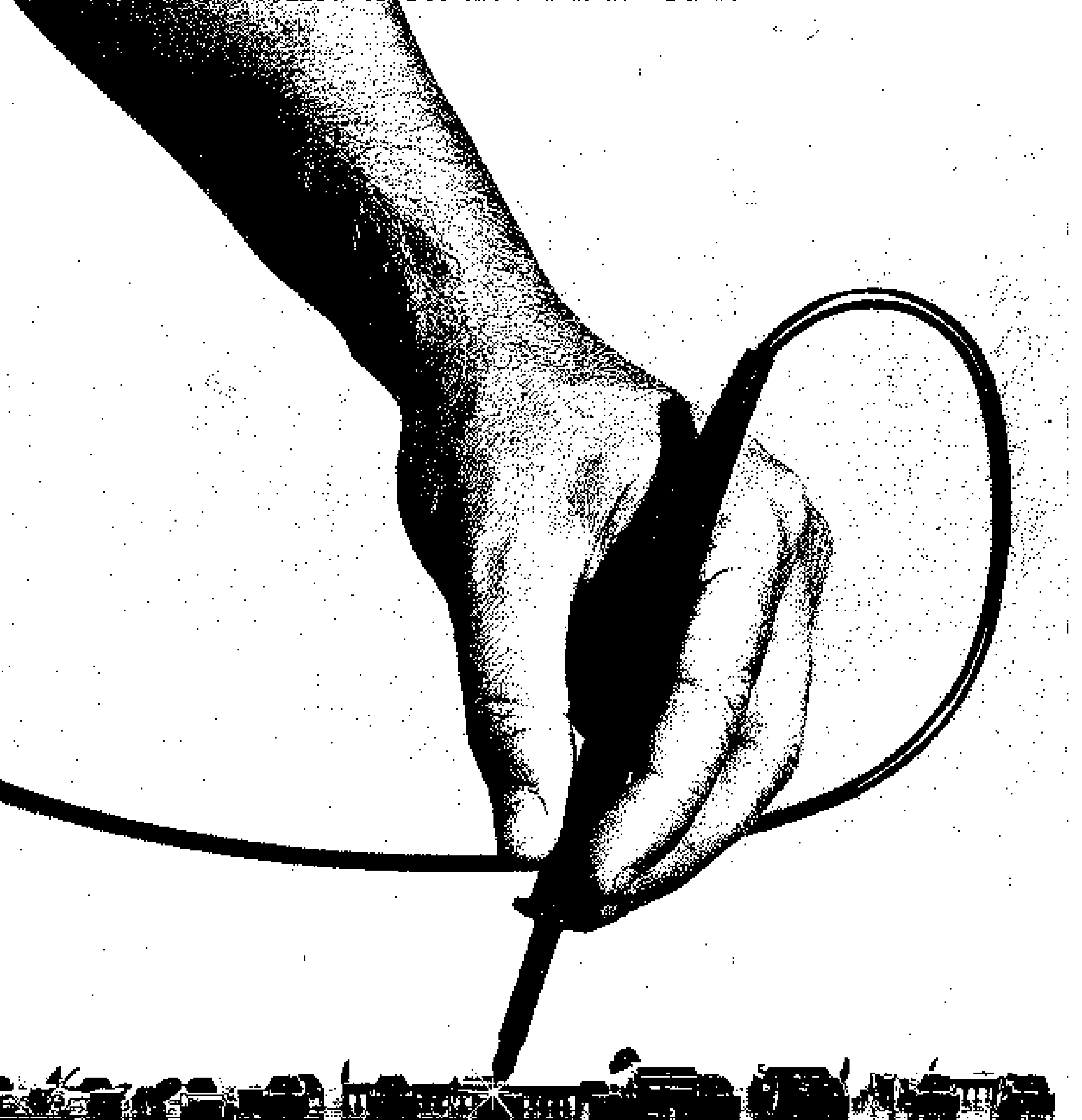
FLUKE®

Technical Data

Application Information B0138

How to Use the 9010A for Guided Fault Isolation



FLUKE®

Introduction

The 9010A Micro-System Troubleshooter makes fault isolation on micro-system boards both simple and easy. The reason is that its built-in test and troubleshooting functions allow you quickly to identify bus-related faults. You can also use the 9010's programmable features to troubleshoot circuits both on and off the bus.

For example, with a set of callable programs to perform guided fault isolation (GFI) you can save money by utilizing lower-skilled operators to find faults.

The purpose of this bulletin is to present an approach to performing GFI in a test environment. It outlines some of the more important considerations to the approach. It also presents a set of 9010A programs which you can use in setting up and performing GFI on your UUT (unit under test). Finally, this bulletin explains how you can integrate the programs into your test and troubleshooting situation.

We estimate that the information contained in this bulletin can save you several weeks of program development time. The programs contained herein should work immediately in your application without modification. However, if you have questions or difficulties with any of the items discussed in this bulletin, please refer them to your Fluke Sales Office.

What Is Guided Fault Isolation?

GFI is a procedure that guides an operator step-by-step through troubleshooting a UUT. It requires transfer of a technician's knowledge of a circuit to a programmable instrument or computer so that less skilled operators can do the testing and troubleshooting. Typically, a computerized GFI procedure will prompt the operator to probe a circuit node, stimulate the circuit, take a reading via the probe, then prompt the operator to take some other action based on the results. This is done sequentially for each node to be probed.

A 9010A program can help perform GFI in two ways. First, a GFI step can be incorporated into a program just occasionally, to verify proper circuit operation at a given node. The purpose of this method is to perform testing, not troubleshooting. Second, there can be a program which does nothing but GFI steps to troubleshoot a UUT which failed the test program.

If you are a typical user, you will undoubtedly have use for GFI at one time or another for production or service. There are likely to be times when you will want to rely upon it heavily because it is needed to compare operation of a known-good board to that of a known-bad one.

However, there are several reasons why GFI may not be needed in your application. Extremely simple UUT's may be so easy to troubleshoot that GFI is impractical. Also, if the UUT has a high component count it may be too time-consuming to have the operator probe each and every node. An in-circuit or bed-of-nails tester might be better suited to a task such as initial turn-on of production boards. Further, some situations lend themselves more readily to immediate mode 9010A troubleshooting than to GFI.

How Should I Approach Testing With The 9010A?

There are some important things to consider when you start testing and troubleshooting with the 9010A, especially when using programs. The following paragraphs outline some of these importances and suggest how to avoid related pitfalls.

Troubleshooting Sequence

During troubleshooting, the technician will usually progress in a sequence through the lowest level circuit known to be failing. The sequence would begin with a check of those items which would most catastrophically affect circuit operation. It would end with those least likely to cause the failure, or most difficult to test. The typical sequence might be as follows:

1. Power supplied to the circuit;
2. Clock and timing supplied to the circuit;
3. Other inputs to the circuit;
4. Clock and timing internal to the circuit;
5. Data paths within the circuit

Because the power supply and clock are analog circuits they should be measured with analog instruments before troubleshooting with the 9010A.

Automatic Checks

Nevertheless, the 9010A can grant a degree of confidence that clock and power are good. For example, it notifies you when clock is missing by displaying POD TIMEOUT. It signals absence of power by displaying UUT POWER FAIL, and power more than 10% too high or low by displaying BAD POWER SUPPLY. Additionally, the probe lights indicate whether the logic levels are within 10% of being below .8 Volts (green), above 2.4 Volts (red), or between .8 and 2.4 Volts for more than 100 nanoseconds.

Setup

When using a program to test your UUT you may find that the 9010A issues a POD TIMEOUT message occasionally even though there is nothing wrong with the pod, the 9010A mainframe, or the UUT. This could occur because a direct memory access (DMA) circuit keeps holding the pod processor so that it can't communicate often enough with the 9010A mainframe. It could happen because the UUT clock is too slow to allow proper pod communication with the mainframe. Or it could be that a forcing line (such as HOLD) is holding the processor in a condition that keeps it from running.

To remedy this, you can use SETUP to change the time out parameter to a larger number, and/or disable the relevant processor forcing lines until the timeout messages no longer occur. Then, when saving programs on a tape, the appropriate setup conditions will be saved also.

Usually, it is a good idea not to disable forcing lines without also disabling any circuits or components in the UUT which use the lines in normal operation. The reason is that disabling a forcing line (such as READY or HOLD) may prevent portions of the UUT from being testable because the 9010A cannot respond to the line when it is disabled except while performing RUN-UUT.

Other Test Equipment
Occasionally, components can be marginal or faulty without being detected by the 9010A. For example, the clock frequency could be much too fast or too slow, or the clock could have poor rise and fall times. Or, the voltage within a circuit area could be high enough to trigger the probe, but low enough to cause mysterious failures, and still not be detected at the processor socket.

For these reasons, it is appropriate to have a scope and voltmeter available to verify that voltage and clock characteristics are correct. And, it is usually a good idea to check those characteristics before troubleshooting other circuits.

You might occasionally have trouble identifying a single failing component which is connected to a node common to several other components. An example of this is a circuit which buses several components together. Therefore, it is a good idea to have a current detection probe or low resistance detection instrument, such as the Fluke 8012A, available for use with your 9010A to isolate the component creating the "stuck node" on the bus.

Use of the Probe
It is important to know that the 9010A takes signatures only in synchronization with processor read/write timing: the pod processor's placement of a valid address or data onto the bus. Some UUT circuits are not so synchronized. As an example, DMA circuits use the address and data buses ONLY when the processor is not using them.

However, you can use the 9010A's logic history and event count features to troubleshoot these and other asynchronous circuits. There are estimates that more than 80% of digital failures consist of a node stuck high or low. Therefore, if the 9010A shows a node to have high and low logic history during a test, then you can generally assume that node to be good. And you can use the event count feature to verify proper operation of devices such as switches, relays, and low-frequency (less than 4500 Hz) devices.

How Do I Structure My Programs For GFI?

A Workable Approach
Although there are many ways to structure test and troubleshooting programs, we have found one approach particularly useful. Generally, that approach is as follows:

1. Develop the programs to test the entire UUT.
2. Develop programs which setup and stimulate each failing circuit so that the 9010A can take readings while the operator probes the circuit.
3. Determine the fault troubleshooting tree for the UUT. The tree should direct the program/operator from failure symptoms to the area or component of the circuit to test.
4. Gather probe data at relevant nodes on a known-good UUT as dictated by the troubleshooting tree.
5. Incorporate the probe data as expected results in a test-and-probe program which guides the operator step-by-step through the UUT troubleshooting procedure.

What Test Programs Should Do
The test program should first perform AUTO-TEST (BUS, ROM, and RAM-SHORT TEST). Next, it should set up and test input/output circuits. Finally, it should test circuits which are asynchronous to bus timing. It should ask the operator what circuit to test in cases where the operator may already have determined a failure to exist in a particular circuit such as a display.

If a test fails, or if the operator must probe a circuit as part of a test, then there should be programs available which perform GFI automatically. To accomplish this, a GFI Supervisor (or the main test program) should set parameters into 9010A registers which identify the item to be probed and expected results. Then, it should call

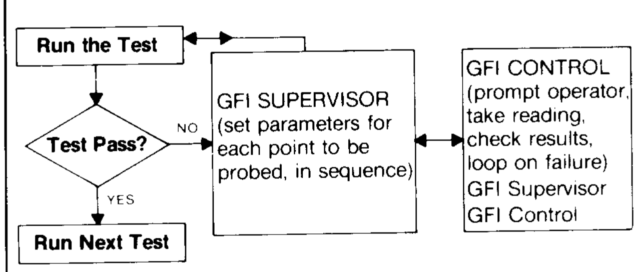


Figure 1. Typical Test and GFI Process

a GFI Control program which uses ancilliary programs as needed to prompt the operator to probe the circuit, take the reading, check it for correctness, and send the pass/fail results to the calling program. It should do this for each point to be probed, thus guiding the operator through the troubleshooting process. Figure 1 illustrates this. It is the same technique we have used in the programs in this bulletin.

Table 1. Typical Test and GFI Program Flow

Main Test Program

1. Display Test Header.
2. Delay to see 9010A display.
3. Display message: "ENTER DESIRED TEST NUMBER" (operator refers to test number list); or start a process of automatically selecting the tests to be performed in a predetermined appropriate sequence.
4. Branch to selected test label.
5. Test label: perform selected test (e.g., the display circuit). Note: this step can have the operator probe a node by performing steps 2a and b of the GFI Supervisor.
6. Did the selected test pass?

   YES: loop to item 3 until all tests have been performed then end.

   NO: execute GFI supervisor program for failing test.
7. End.

GFI Supervisor Program For Failing Circuit

1. Display message: Circuit Test header.
2. For each point to be probed:
   a. Set registers C, 8, and 9.
   b. Execute the GFI Control program to prompt the operator, take the reading, display the results, and loop on failures.
   c. Did operator press the CLEAR key?

      NO: Loop to item 2 to probe next point.

      YES: End the GFI activity.

The GFI programs are set up so that your test program (or the GFI Supervisor) must set probe information parameters into registers C, 8, and 9 before calling the GFI Controller to enable the operator to probe a point. The GFI Controller uses the parameters to determine display prompts and expected readings for that point. The GFI programs leave these registers unchanged, but do modify all the other global registers. Subsequent paragraphs describe the required contents of the registers. Your program flow might resemble that in Table 1.

How Does The 9010A Perform Guided Fault Isolation?

The 9010A uses its probe and synchronization capabilites with the READ PROBE function to determine what logic activity has occurred at the point being probed. The READ PROBE operation allows the 9010A to glean logic level history, event counts, and signatures from the circuit as described in the 9010A operator manual. The 9010A programs in this bulletin use these capabilities to do the following:

1. Write data to relevant addresses as necessary to enable devices in the circuit being probed. An example of this is to write data to an input-output circuit such as a Programmable Interface Adapter chip (PIA) to set up its registers for either input, or output, or both.
2. Instruct the operator to place the probe on a particular component lead in the circuit, and continue only when the operator has done so.
3. Select the appropriate synchronization mode (Address, Data, or Free-run), and perform a READ PROBE operation. This clears the present probe information and initiates the 9010A process of monitoring the logic activity at the probe tip.
4. Write or read data to/from the circuit being checked as necessary to stimulate it. In the case of address and data synchronization modes, this allows the probe to gather data during operations which are synchronous to Address and Data valid periods on the processor bus. This step may not be necessary in the case of timing, direct memory access (DMA), and certain other circuits because they are asynchronous to or operate independent of active microprocessor timing.
5. Perform a READ PROBE to complete the data gathering.
6. Separate the fields of register 0 to isolate the signature, event count, or logic level history, then compare one or more of them to a predetermined, known-good value.

FLUKE®

7. Display the results for the operator, and give the operator such options as aborting the test, looping on the failure, or skipping to the next point to be probed. During looping, the operator can heat or cool the circuit, flex the board, or do other actions to discover the cause for intermittent failures.
8. Inform the operator of one or more suspected bad components in the event of a solid failure.
9. Continue with the preceding steps through the entire circuit until the program has isolated the failing component.

Program Examples

Tables 2 and 5 list a universal set of 9010A programs which you can incorporate into your 9010A troubleshooting scheme to perform guided fault isolation. The programs are generic in that they can be used for almost any UUT. They use about 3000 bytes, leaving plenty of room for your test programs. The programs can be modified easily if changes are needed for your application. The remainder of this bulletin explains the programs in detail and describes how to use them in your troubleshooting environment.

We have used several small programs to perform the probe-and-test function for GFI. Each program performs a specific function. This makes the GFI program set modular and much more easily readable than would be a large, monolithic program. You will have to write one or more programs of your own to supervise the GFI activity. You may use any otherwise unused program numbers for them; our sample program listings use program number 20 with the name "GFI Supervisor."

Before developing program 20, you will be running program 22 to get readings from a known-good UUT and display parameters for registers C, 8, and 9. In developing program 20, you will enter steps which preset those parameters into the registers and execute program 21. Program 21 will use the parameters to determine how to test the circuit. A subsequent topic describes how the parameters are structured and how to derive them. Once you have taken all the known-good readings needed to troubleshoot your UUT, you can delete program 22 and grant yourself another 1379 bytes of memory.

Table 2. Summary of Guided Fault Isolation Programs

Number	Name And Description
0	**MAIN TEST** - Your program to perform a functional test of your UUT. When it fails, it should give the operator the option of performing GFI. It would call program 20 and others like it to do the GFI. Or, it would contain the functions of program 20 itself.
3	**KEY ENABLE** - Toggles the enabled/disabled state of the asynchronous keyboard interrupt for programs 4, 21, 22, and 24.
4	**KEY WAIT** - Halts program 22 until the operator presses any key except STOP.
9	**DELAY** - Delays the program to give the operator time to read the display in programs 21 and 22.
20	**SUPERVISOR** - Supervises the flow and sequence of the GFI tests. This program sets up parameters in registers to identify: which programs to execute to setup and stimulate the circuit under test; the point to be probed; whether to check signature, count, or logic history; what sync mode to use; and up to two suspected faulty components. It then calls program 21 for each point to be probed to control the test.
21	**CONTROLLER** - Examines parameters passed by program 20 to: execute the UUT setup program; display point to be probed and expected results; initiate the probing operation; compare actual and expected results; allow looping on failure; display suspected faulty components.
22	**PACKER** - (Not used in the final program) - Prompts operator for information about the node to be probed; stimulates and gathers data from a known-good circuit under test just as program 21 would in an actual GFI situation; displays relevant parameters for registers C, 8, and 9 to be keyed into GFI Supervisor program 20.
23	**READER** - Uses parameters in registers C and 9 to select the sync mode (address, data, free-run), call the program to stimulate the circuit under test, and take the selected type of reading (signature, logic level history, or event count) for programs 21 and 22.
24	**MONITOR** - Performs READ-PROBES in free-run sync for program 21 to determine whether the probe is touching the circuit. Loops continuously until 10 successive readings indicate the probe is properly placed. Used to detect probe in the circuit, and out of the circuit.
25	**D-HISTORY** - Displays expected and actual logic history for programs 23, 26.
26	**D-EXPECTED** - Displays test mode (signature, history, count) and expected results for programs 20 and 22.
27	**D-DEVICE** - Displays one of 15 types of components (U, R, C, etc.) for programs 21 and 22. Used for device to be probed and suspected bad devices.
n	**SETUP/STIMULUS** - Setup/stimulate the circuit under test for probing. Program numbers (any unused ones of your choosing) are set in register C by program 22 and 21. Technician writes them (not provided in this bulletin).

FLUKE®

Figure 2 shows the basic interaction of the GFI programs. It does not show the delay routines because they are inconsequential to the flow. It also does not show program 22 because it is not used during normal GFI testing.

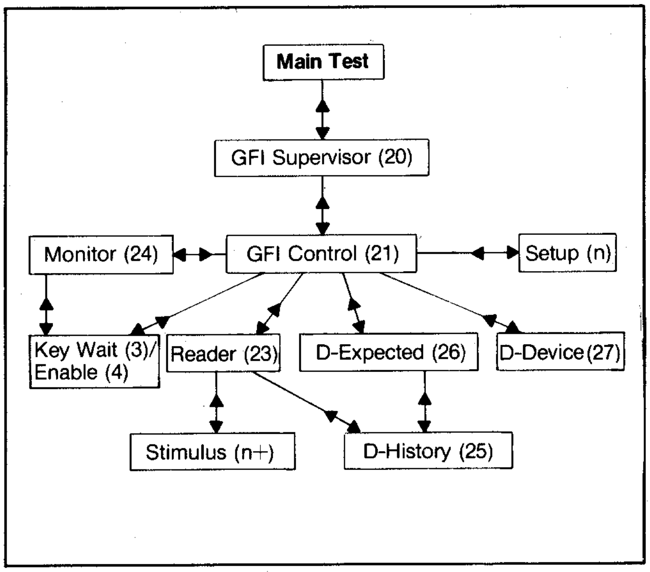


Figure 2. Interaction of GFI Programs.

How Do I Set Up To Use The Programs?

Step One: Configure the System

The only materials needed to perform GFI on your board are the 9010A with pod and probe, your known-good unit under test (UUT), and the GFI programs on a 9010A tape. The 9010A, pod, probe, and your UUT must be configured as described in the 9010A operator manual. Refer to that manual as necessary to prevent causing damage to the 9010A, the pod, or your UUT.

Step Two: Store the GFI Programs into 9010A Memory

To store the GFI programs into your 9010A, you can key them into your 9010A. However, to save time you can load them in from the pre-recorded Guided Fault Isolation cassette tape, available through your Fluke Sales Office. If you decide to key them in, compare them line-by-line with the listings to ensure they are correct.

Step Three: Write Programs to Setup and Stimulate the Circuit Under Test

The GFI programs allow you to specify two program numbers in your Main Test or Supervisor program, a setup program number, and a stimulus program number. The setup program sets up conditions for the circuit under test. For example, it might configure an input/output circuit such as a PIA to transfer data to a peripheral device.

The stimulus program stimulates the circuit under test while the operator is probing the circuit to find faults. For example, once the setup program has initialized the PIA, the stimulus program might perform write operations to send data through the circuit.

As another example, the setup or stimulus program might stop the test and prompt the operator to set a switch or perform some other action, then to press CONT when ready to proceed. This grants your GFI activity the flexibility needed for complete operator and UUT interaction.

There are cases where you will not need to specify one or both of these programs. An example would be a circuit which is always passing timing signals without being setup or stimulated by the 9010A. In this case, the GFI routine can simply look for high and low logic activity at nodes in the circuit without performing a setup or stimulus program.

Step Four: List Information About the Points to Be Probed

After a study of the circuit and your testing approach, you will want to write a list of the circuit components you wish to probe during the test. The GFI programs will need that and other information as follows:

1. The component type and number (reference designator, such as U12 for integrated circuit #12, SW2 for switch #2, etc.), and pin number to probe. Program 27 allows you to specify up to 15 types of devices to be probed or identified as suspected bad devices. You may modify the program to allow others of your choosing (see the program 27 listing). Currently, 13 devices are implemented. They are: U (integrated circuit), Q (transistor), R (resistor), C (capacitor), CR (diode), SW (switch), LED (light emitting diode), KEY (pushbutton or key), K (relay), P (plug), J (jack), X (component socket), and BP (backplane).
   These mnemonics are for display nomenclature only, so that nothing prevents you from calling an integrated circuit a resistor.
2. The first and second components (type and number) you suspect to be bad if the test fails.
3. Whether to take signature, logic history, or event count, and whether to use Address, Data, or Free-run sync.
4. What programs to execute in order to set up the circuit to be tested and to stimulate the circuit during the data gathering process.

Step Five: Run PACKER Program 22

Program 22 prompts you to enter information from the list in step four, sets up and stimulates the circuit while gathering the selected information, then packs the information together and displays appropriate contents for registers C, 8, and 9. It also sends the register values to the RS-232 interface for printing or display on a terminal.

The program displays prompts for operator actions as follows:

1. **Prompt:** PARAMETER PACK PROGRAM (display Program Reader)
   Action: none
2. **Prompt:** DEVICE TO PROBE <1-F,ENTER>
   Action: Press a hex key 1 through D to select the type of component on which to place the probe tip, then press ENTER. If you make a mistake or wish to view the optional selections, simply press another hex key before pressing enter. Key entries are: 1=U, 2=Q, 3=R, 4=C, 5=CR, 6=SW, 7=LED, 8=KEY, 9=K, A=P, B=J, C=X, D=BP. Zero is not allowed; E and F are, but are currently unused.
3. **Prompt:** ENTER DEVICE NUMBER <256 = _
   Action: Enter a decimal value from 0 through 255 as the number of the device to probe. The GFI programs will use this to display the reference designator of the probed device, such as U21, J5, etc.
4. **Prompt:** ENTER PIN NUMBER <256 = _
   Action: Enter a decimal value from 0 through 255 as the pin number of the device to be probed. The GFI programs will use this to display the reference designator's pin number to be probed, such as U21 PIN 7, etc.
5. **Prompt:** 1ST SUSPECT DEVICE TYPE <0-F,ENTER>
   Action: Press a hex key to select the type of component you most suspect is the cause of a failure at the node described by prompts 1-3, then press ENTER. The code types are the same as for prompt 2. If you press ENTER only, the program will assume there are no suspect devices and advance to prompt 9.
6. **Prompt:** 1ST SUSPECT NUMBER <256 = _
   Action: Enter a decimal value from 0 through 255 as the number of the device in prompt 5. The GFI programs will use this to display the reference designator of the first suspected bad device, such as U21, R5, etc.
7. **Prompt:** 2ND SUSPECT DEVICE TYPE <0-F,ENTER>
   Action: Press a hex key to select the device type for the next component you most suspect is the cause of a failure at the node described by prompts 1-3, then press ENTER. The code types are the same as for prompt 2. If you press ENTER only, the program will assume there is no second suspect device and advance to prompt 9.
8. **Prompt:** 2ND SUSPECT NUMBER <256 = _
   Action: Enter a decimal value from 0 through 255 as the number of the device in prompt 7. The GFI programs will use this to display the reference designator of the second suspected bad device, such as U23, C7, etc.
9. **Prompt:** PRESS 0=SIG, 1=LEVEL, 2=COUNT
   Action: Press 0 for taking signatures, 1 for logic level history, or 2 for event counts. There is no need to press ENTER. The GFI programs will use this to select the reading mode for gathering selected read-probe information from the point being probed. If you press 2 the program will force FREE-RUN sync mode (see prompt 10) and advance to prompt 11.
10. **Prompt:** PRESS 0=FREE 1=ADRS 2=DATA SYNC
    Action: Press 0, 1, or 2 to enable free run, address, or data sync mode. There is no need to press ENTER. The GFI programs will use this to set the sync mode for gathering read-probe information from the point being probed. Selecting free run sync along with signature readings is not allowed and will return you to prompt 9.
11. **Prompt:** SETUP PGM = _
    Action: Enter a decimal value from 0 to 99 (e.g., 50) as the number of the program to be executed to set up the UUT circuit being tested. If you press ENTER only or enter a value of 0, then you are specifying that there is no setup program. If the program does not exist, the PACKER program will later abort with FATAL-PROGRAM NOT FOUND message. You will need to create the missing program and re-run program 22.
12. **Prompt:** SETUP PGM=50, STIMULUS PGM=_
    Action: Enter a decimal value from 0 to 99 as the number of the program to be executed to stimulate the UUT while the operator is probing the circuit being tested. If you press ENTER only or enter a value of 0, then you are specifying that there is no stimulus program. If the program does not exist,

the PACKER program will abort with FATAL-PROGRAM NOT FOUND message. You will need to key in the missing program and re-run program 22.

If you specify no stimulus program, then the actual GFI Controller will perform two immediately successive READ-PROBE operations (about 20 milliseconds apart) without exercising the UUT. This might be useful for monitoring event counts or logic history at node which has a continuous stream of pulses.

13. **Prompt:**

U21-7 COUNT = cc, CONT	displays count
or U21-7 SIG = ssss, CONT	displays hex signature
or U21-7 LEVEL = hxl, CONT	displays logic level history
or U21-7 LEVEL = NONE, CONT	

Comment: The program has executed the setup program if specified, and is running a loop which executes the stimulus program (if specified) for the known-good UUT while you are probing the specified component in the circuit. The display shows the reading taken. This allows you to see correct readings from a known-good UUT before performing GFI on a suspected bad UUT.

Action: Press CONT to stop looping and proceed to prompt 14, or CLEAR to return to prompt 9 and select different stimulus/exercise programs and sync/reading modes. ***DO NOT*** press CONT nor remove the probe unless one of the above messages is being displayed, signifying that the program has stored a valid reading. The reading will be needed in prompt 14.

At this point you can probe other components in the circuit as well as the one displayed. History and signature readings are only valid if they are stable and thus predictable. Therefore, let the program loop a few times.

If you are taking signatures and they are not stable, then the logic activity at the point being probed is asynchronous to microprocessor bus timing. You will either have to change the stimulus program or check for logic history or event counts rather than signatures at that node.

The count can represent more than 128 events because the 128-event counter is circular. Between two successive read-probe operations, it will overflow at frequencies above about 4500 Hertz. Also, variations in frequency of the 9010A crystal and the probed signal, and the 20-millisecond window between successive read-probes, combine to vary the displayed count, especially when you probe a high frequency signal asynchronous to bus timing. Therefore, use event counts only for very low frequency asynchronous signals, or for any signals which are synchronous to bus timing or can produce a controlled, predetermined number of events.

The GFI programs will allow a test to pass if the UUT count is within a specified range. Therefore, when observing hex counts during prompt 13, record the minimum and maximum counts displayed. Prompt 14 requests them. Usually, the range is small, such as 37 minimum to 42 maximum. When near the counter wrap point, the range will seem large (such as 3 minimum to 7C maximum) when it is actually small (7C minimum to 3, actually 83 hex, maximum).

Although the range could actually be large, usually you should record the higher value as the *minimum* and the lower value as the *maximum*. The test will pass if the reading is between the minimum and the maximum, inclusive.

14. **Prompt:**

ENTER SIGNATURE ssss = _
  display last signature read
or ENTER LVL <4,2,1=L,X,H> v = _
  display code for last level
or ENTER COUNT MIN nn = _
  display lowest count read
then ENTER COUNT MIN nn = 23, MAX xx= _
  display highest count read

Comment: One of these messages is displayed in response to pressing CONT at prompt 13, depending on the type of reading taken. The message is prompting you to enter the last value read during the loop. The program saved it, assuming it was stable.

Action: For signatures or level, press ENTER only to use the last value read. For count min, press ENTER only to use the lowest count read during the loop taking readings. For count max, press ENTER only to use the highest count read. Otherwise, key in the desired value and press ENTER. Use hex for signature and level; decimal for counts.

If you enter a value in excess of FFFF for a signature, 7 for level history, or 127 for count, then the program will use that entry as the last value read, destroying the actual last value read, and will request the entry anew.

When entering the code for logic level, use the sum of the bit values: High=1, Invalid[X]=2, Low=4. For example, if Low and Invalid are read, the code is 4(Low) + 2(Invalid)=6.

Table 3. Format for Registers C, 8, and 9 Before Executing Program 21

Note: all values are shown in hexadecimal form. Program numbers, device numbers, and pin numbers are decimal values converted to hex.

Register C = PPSS, where
PP = setup program number
SS = stimulus program number

Register 8 = VVVVDDPP, where
VVVV = the expected value in the reading, formatted as follows:
SSSS = the signature, or
NNXX = the minimum (NN) and maximum (XX) counts, or
0L00 = the logic level history bits as follows:
0LXH = bits set to 1 for Low, invalid (X), High
DD = the device number of the device to be probed
PP = the pin number to be probed on that device

Register 9 = ISSJFFKM, where
I = the device type code for the second suspect bad component
SS = the device number of the second suspect bad component
J = the device type code for the first suspect bad component
FF = the device number of the first suspect bad component
K = the device type code for the device to be probed
M = the sync and reading mode bits DACH, formatted as follows:

D = 1 for data sync	C = 1 for event counts
A = 1 for address sync	H = 1 for logic history
DA = 0 for free-run	CH = 0 for signatures

15. **Prompt:** REG C=nnnn, 8=nnnnnnnn, 9=nnnnnnnn
Action: Record the values shown on the display because they are register parameters for you to use when developing the GFI supervisor or main test program. Once they are recorded, press CONT to start over at prompt 2 to gather information and build parameters for the next test point. Table 3 gives the format for the parameters in registers C, 8, and 9. As an example, if you wanted to probe U5 pin 27 for signature 6B3F in data sync mode using setup program 67 and stimulus program 81, and suspected CR115 or U26 bad in the event of a bad signature, the register parameters would be:

Reg C = 4351, Reg 8 = 6B3F051B, Reg 9 = 11A57318.

If you have a terminal or printer properly interfaced to the 9010A via its RS-232 port, then the parameters will be displayed or printed for you, saving you the trouble of writing them.

You may wish to hand-enter parameters which change only slightly from test point to test point. For example, you may expect to get the same reading on pin 82 as you got on pin 27. Since device and pin numbers must be in hexadecimal form, you will need to convert the decimal pin numbers to hex before recording them. Program steps to convert decimal to hex and vice versa are as follows:

Table 4. Typical GFI Supervisor Program Sequence

1. LABEL 1	Test first node
2. REGC = 4351	Set register C parameter
3. REG8 = 6B3F0 51B	Set register 8 parameter
4. REG9 = 11A57318	Set register 9 parameter
5. EXECUTE PROGRAM 21	Call GFI control: take and compare reading
6. IF REGB = 1D GOTO F	End if CLEAR key pressed
7. IF REGB = 26 GOTO 1	Repeat check if RPEAT pressed
	Perform 1-7 for each node to be probed
LABEL F	End of program

DPY-DECIMAL 1=$1 HEX converts decimal to hex
DPY-HEX /1=@1 DECIMAL converts hex to decimal

Step Six: Enter the Parameters Recorded into SUPERVISOR Program 20

Now you need to enter the parameters recorded from step five into registers C, 8, and 9 via program 20 or your main test program. Table 4 gives a typical sequence for doing this. You must enter program steps 2 through 5 for each node to be probed, omitting registers whose values do not change for the next node. You can enter steps 2-5 in the appropriate spots in the main test program rather than in a separate GFI Supervisor.

Consider these important points when creating the Supervisor program:

1. Although the Controller attempts to enable the asynchronous key interrupt, it actually only toggles the interrupt from its current condition. Therefore, your test program or the Supervisor must have the interrupt disabled before executing the Controller.

2. The Controller communicates with the Supervisor by loading the B register with the value of the CLEAR key if the operator presses CLEAR when the test is looping or in response to the CONT? prompt when suspected bad components are displayed. The B register contains the value of any other key pressed when the STOPPED light is off and the Controller is not looping on a failure (see next topic). Therefore, the Supervisor must use key codes in the B register to control flow of the GFI activity. An example of this is

shown in Table 4. The Supervisor terminates if the operator presses CLEAR, and repeats if RPEAT.

3. The Controller loads the last reading taken into register E before exiting to the Supervisor. Signature or count will be in the lower bits; History will be in bits 24, 25, and 26, as they would be in register 0 after a READ PROBE. This can be useful for further program manipulation of the reading.

Step Seven: Run the SUPERVISOR Program 20

Now, whenever your main test program fails, it can execute program 20 to determine the cause of the failure. Program 20 (or your main test program) will execute program 21 which instructs the operator to probe a point, waits until the probe is in place, takes the reading, compares it to a known-good one, loops on a failure if desired, and displays the suspect components if not. This is described in detail in the next topic.

How Do I Operate The SUPERVISOR?

Once called by your main test program or program 20, the GFI Controller (21) displays only the five types of prompts for operator actions given in the following paragraphs. For more detail on the flow of the GFI programs while the test is running, refer to Figure 3.

1. **Prompt:** PROBE U5 PIN 12
   Action: The operator must touch the probe to the circuit at U5 pin 12 and hold it there. The program senses that the probe is in place after about one second, then sets up and stimulates the UUT circuit, takes the reading, and compares the actual result to the expected result. If the reading is good, then the program displays prompt 5. If not, it displays prompt 2.

   If nothing happens after a few seconds, then the operator should assume that the circuit is dead and press CONT. The program will assume a failure and display the BAD message in prompt 2. If the operator presses any other key during the one-second period, then the program will advance to prompt 1 for the next test point without taking a reading for the U5 pin 12.

   If the program expects a reading of Invalid logic level history only, then it will append ", CONT" to the prompt 1 display and not try to sense the presence of the probe in the circuit. In this case, the operator must press CONT to cause the program to start taking the reading.

2. **Prompt:** U5-12 CNT 10-20 = 5, BAD, LOOP?
   Action: The operator should press YES to loop on the failing test, or NO to display the suspected bad components. The display shows a test failure because a count of 10 through 20 was expected, but only 5 was received.

3. **Prompt:** U5-12 CNT 1 0-20 = 5, BAD, CONT
   (display failure in loop)
   or U5-12 CNT 10-20 = 15, GOOD, CONT
   (display pass in loop)
   Comment: The program will display one of these types of messages while looping on a failure (you pressed YES in response to prompt 2)

   Action: The operator should press CONT to stop the loop, CLEAR to abort the test program altogether, or any other key to advance to prompt 1 for the next test point. If the operator presses CONT, then the program will take a final reading and go to prompt 2 or 5.

   If the GOOD message appears occasionally amongst BAD ones, the operator can assume the failure is intermittent. It is possible that there is no failure, but that the GFI Supervisor passed an erroneous parameter, or the programmer asked for event counts when logic history should have sufficed on a severely unstable mode.

4. **Prompt:** SUSPECT BAD U5, Q12, CONT?
   (display suspect components)
   Action: The operator should press YES to advance the program to prompt 1 for the next test point, or NO to terminate the test program altogether. The program advanced to this prompt when the operator pressed NO in response to prompt 2.

5. **Prompt:** U5-12 CNT 10-20 = 15, GOOD
   Action: No operator action. The program displays this message to notify the operator that the test passed from prompt 1 or passed from prompt 3 after the operator pressed CONT. The display will be visible only momentarily, then the program will advance to prompt 1 for the next test point.

Conclusion

The guided fault isolation programs in this bulletin should work well in your application with little or no modification. Further, a good understanding of their philosophy, nature, and flow should help you simplify the task of testing and troubleshooting your microprocessor-based systems.

Figure 3.. Functional Flow of GFI Programs

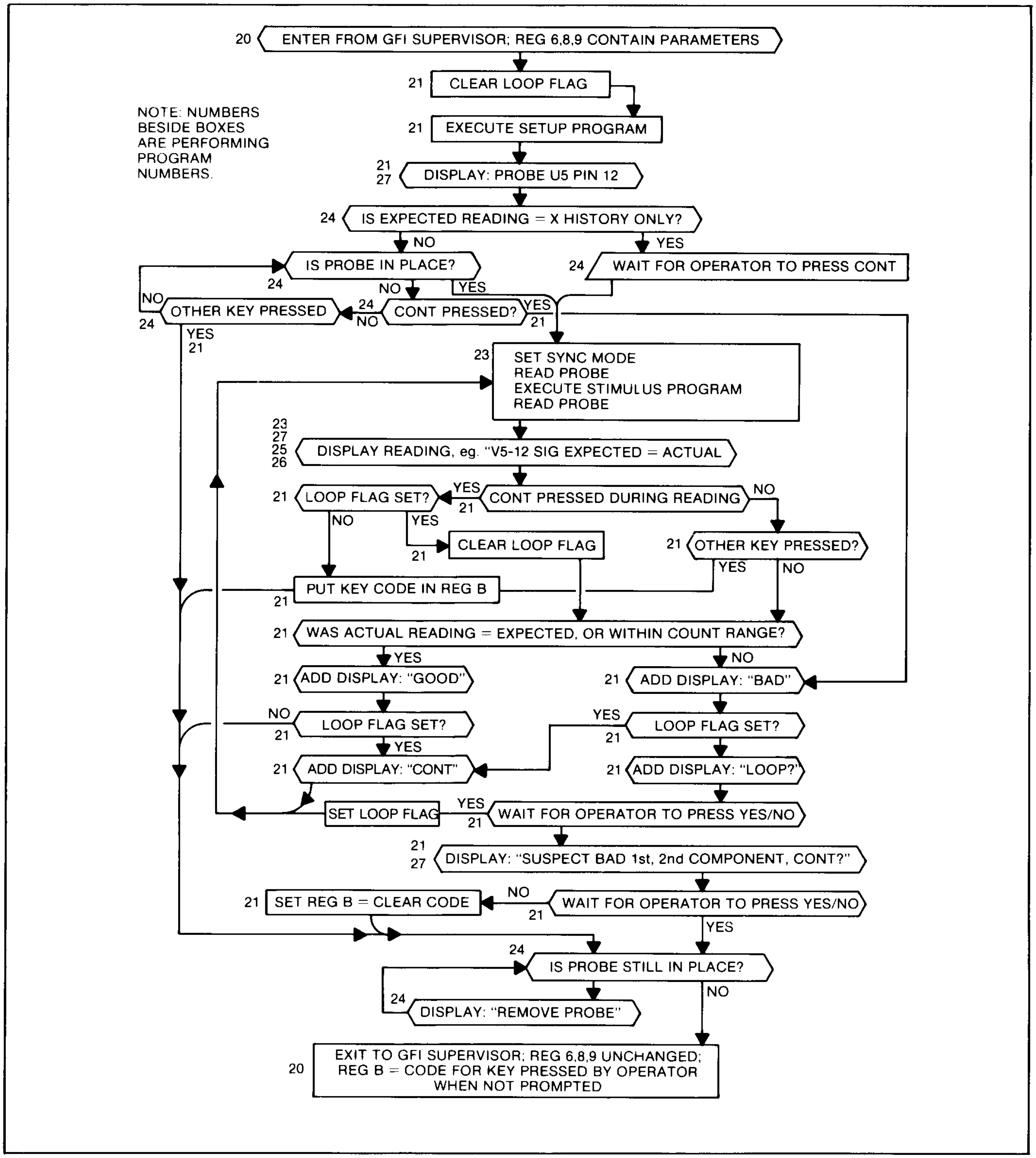


Table 5. Guided Fault Isolation Program Listings

```
PROGRAM 3   KEY ENABLE  12 BYTES            Enable key interrupt

Inputs:      none
Called by:   programs 4, 21, 22, 24
Calls to:    none
Output:      Reg B = 40; toggles the enabling of asynchronous keyboard interrupt

   REGB = 40                                Initialize reg B
   DPY-+XB                                  Enable key interrupt to reg B

PROGRAM 4   KEY WAIT  18 BYTES              Wait for key interrupt

Inputs:      none
Called by:   program 22
Calls to:    program 3
Outputs:     Register B = the value of a key pressed (0-3F)

   EXECUTE PROGRAM 3                        Enable interrupt
1: LABEL 1
   IF REGB = 40 GOTO 1                      Loop till key pressed (<40)

PROGRAM 9   DELAY  28 BYTES                 Delay loop (variable time)

Inputs:      Register F = delay loop parameter
Called by:   programs 21, 22
Calls to:    none
Output:      delay time to delay program execution for displays

   REG1 = REGF                              Save delay parameter
0: LABEL 0                                  Loop reg 1 times
   IF REG1 = 0 GOTO 1                       Exit if end of loop
   DEC REG1
   GOTO 0
1: LABEL 1                                  End loop

PROGRAM 20   SUPERVISOR  26 BYTES           Main Guided Fault Isolation program

Inputs:      Reg B = any key pressed during Controller program when not looping
             and not stopped.
Called by:   any main test program upon detecting failure
Calls to:    program 21
Outputs:     Registers C, 8, and 9 as given by program 22

   DPY-GFI SUPERVISOR PROGRAM               Enter your GFI program here

PROGRAM 21   CONTROLLER  609 BYTES          Controls the GFI activity

Inputs:      Registers C, 8, and 9 as specified by program 20
Called by:   program 20 or other test program
Calls to:    programs 3, 27, 24, 26, 23, 9, Setup program specified by reg C
Outputs:     Reg B = value of key pressed by operator during program; reg E =
             last reading; reg C,8,9 stay unchanged; other globals may change

   IF REGC AND FF00 = 0 GOTO 0              Branch: no setup program number
   REG3 = REGC SHR SHR SHR SHR SHR SHR      Get setup program number
SHR SHR AND FF
   EXECUTE PROGRAM REG3                     Condition UUT with setup program
0: LABEL 0
```

```
    REG3 = REG8 AND FF                          Get pin number to probe
    REGA = REG8 SHR SHR SHR SHR SHR SHR         Setup to get component number
SHR SHR
    REG2 = REGA AND FF                          Get component number to probe
    REG4 = REGA SHR SHR SHR SHR SHR SHR         Get signature/lo-hi/history
SHR SHR
    REG5 = 0                                    Initialize loop flag to no loop
    REG6 = REG9 SHR SHR SHR SHR                 Get component types
    REG7 = REGC                                 Save setup/stimulus pgm # parameter
    EXECUTE PROGRAM 3                           Enable key interrupt for exit
    DPY-#PROBE _                                Display message to probe
    REGA = REG6                                 Set global:component type parameter
    EXECUTE PROGRAM 27                          Display component type to probe
    DPY-+@2 PIN @3                              Display chip & pin # to probe
    REGA = 0                                    Set global: wait-for-probe flag
    EXECUTE PROGRAM 24                          Wait till probe is in place
    IF REGB = 40 GOTO 1                         Branch: operator didn't press key
    IF 40 > REGB GOTO F                         Exit: operator pressed key not CONT
    REGB = 40                                   Pressed CONT:Reset key interrupt
    GOTO 5                                      FAIL: no circuit activity at probe
1:  LABEL 1                                     Loop point for checking circuit
    DPY-                                        Clear display
    REGA = REG6                                 Set global:component type parameter
    EXECUTE PROGRAM 27                          Display component type being probed
    DPY-+@2-@3                                  Display chip & pin # being probed
    REGA = REG4                                 Set global: expected result
    EXECUTE PROGRAM 26                          Display expected result
    REG0 = REGA                                 Get expected sig/history/min count
    REG1 = REGD                                 Get max count
    REGC = REG7                                 Restore setup/stimulus pgm parametr
    EXECUTE PROGRAM 23                          Set sync mode, stimulate, rd probe
    REGE = REGA                                 Save reading for use by Supervisor
    IF 40 > REGB GOTO A                         Branch: operator pressed key
2:  LABEL 2                                     Loop point for CONT during loop
    IF REG9 AND 2 > 0 GOTO 3                    Branch: check for correct count
    IF REGA = REG0 GOTO 8                       Branch: sig/history is correct
    GOTO 5                                      Branch: bad signature/logic history
3:  LABEL 3                                     Check for correct count
    IF REG0 > REG1 GOTO 4                       Branch: min is > max (count wrap)
    IF REGA > REG1 GOTO 5                       Branch: actual is > max (bad)
    IF REG0 > REGA GOTO 5                       Branch: actual is < min (bad)
    GOTO 8                                      Branch: count within min-max (good)
4:  LABEL 4                                     Count wrap limit check
    IF REG1 >= REGA GOTO 8                      Branch: actual is < max (good)
    IF REGA >= REG0 GOTO 8                      Branch: actual is >=min (good)
5:  LABEL 5                                     Handle bad/no readings
    IF 40 > REGB GOTO A                         Branch: operator pressed key
    DPY-+BAD                                    Display failure message
    IF REG5 > 0 GOTO 9                          Branch: loop if loop flag is set
    DPY-+#                                      Beep
    REGF = 2                                    Set Delay parameter
    EXECUTE PROGRAM 9                           Delay to hear second beep
    DPY-+#; LOOP?5                              Ask whether to loop on failure
    IF REG5 > 0 GOTO 1                          Branch: loop flag is now set
    DPY-SUSPECT BAD _                           No loop: display message
    REGC = REG7                                 Restore setup/stimulus pgm #s
6:  LABEL 6                                     Loop to display suspect components
    REG6 = REG6 SHR SHR SHR SHR                 Get suspect component #
    REG7 = REG6 AND FF                          Mask out other info
    IF REG7 = 0 GOTO 7                          Branch: endloop if no suspect
    REG6 = REG6 SHR SHR SHR SHR SHR SHR         Get suspect component type
SHR SHR
```

FLUKE®

```
   REGA = REG6                              Set global: suspect type
   EXECUTE PROGRAM 27                       Display suspect component type
   DPY-+@7,_                                Display suspect component #
   GOTO 6                                   Branch: loop to get next suspect
7: LABEL 7                                  No more suspects:
   DPY-+#CONT?6                             Ask whether to continue GFI
   IF REG6 = 1 GOTO A                       Branch: get next point if yes
   IF 40 > REGB GOTO C                      Branch: operator pressed key
   DPY-+%B                                  Disable key interrupt
   GOTO C                                   Branch to exit (no continue)
8: LABEL 8                                  Comparison was good
   DPY-+GOOD                                Display good message
   IF 40 > REGB GOTO A                      Branch: operator pressed key
   IF REG5 = 0 GOTO A                       Branch: loop flag is clear
9: LABEL 9                                  Loop flag is set:
   DPY-+;CONT                               Display CONT message
   REGF = 25                                Set delay parameter
   EXECUTE PROGRAM 9                        Delay to see message
   GOTO 1                                   Loop to beginning
A: LABEL A                                  Enable exit:
   REGF = 25                                Set delay parameter
   EXECUTE PROGRAM 9                        Delay to see reading
   REGC = REG7                              Restore setup/stimulus pgm #s
   IF REGB = 40 GOTO D                      Branch: operator didn't press key
   IF REGB = 25 GOTO B                      Branch: CONT pressed
   GOTO F                                   Branch: other key pressed
B: LABEL B                                  CONT was pressed
   REGB = 40                                Reset interrupt register
   IF REG5 = 0 GOTO F                       Branch: Loop flag is clear
   REG5 = 0                                 Clear loop flag
   DPY-+%B                                  Re-enable key interrupt
   GOTO 2                                   Loop to re-display readings
C: LABEL C                                  Wait-for-probe loop failed
   REGB = 1D                                Set clear code into interrupt reg
   GOTO F                                   Branch: exit
D: LABEL D                                  No key pressed during routine:
   DPY-+%B                                  Disable key interrupt
F: LABEL F                                  Exit:
   REGA = 1                                 Set global: remove-probe flag
   EXECUTE PROGRAM 24                       Wait for probe to be removed
```

PROGRAM 22   PACKER   1379 BYTES   Packs parameters for reg C, 8, 9

```
Inputs:     none
Called by:  none (standalone)
Calls to:   programs 3, 4, 9, 23, 26, 27, Setup and Stimulus Programs
Outputs:    registers C, 8, and 9 for use by program 20
```

```
   DPY-PARAMETER PACK PROGRAM#              Display message
   REGF = 50                                Set delay parameter
   EXECUTE PROGRAM 9                        Delay to see display
   REGA = 0                                 Set global: clear type parameter
0: LABEL 0
   DPY-DEVICE TO PROBE (1-F,ENTER)          Ask for device type
   DPY-+___
   EXECUTE PROGRAM 27                       Display device type
   EXECUTE PROGRAM 4                        Wait for operator to press key
   REG0 = REGA                              Save device type
   REGA = 0                                 Set global: clear type parameter
   IF REGB = 1C GOTO 1                      Branch: ENTER key was pressed
   IF REGB = 0 GOTO 0                       Branch: 0 type not allowed
   REGA = REGB                              Set global: save device type
   IF F >= REGB GOTO 0                      Branch: 1-F types allowed
   REGA = REG0                              Illegal type: restore last good one
   DPY-+#                                   Beep for erroneous entry
```

```
    GOTO 0                                  Branch: loop till ENTER pressed
1:  LABEL 1
    DPY-ENTER DEVICE NUMBER <256 =          Ask for device number
    DPY-+ \7                                Save in reg 7
    IF REG7 > FF GOTO 1                     Branch: >FF not allowed
2:  LABEL 2
    DPY-ENTER PIN NUMBER <256 = \6          Ask for pin numberl; save in reg 6
    IF REG6 > FF GOTO 2                     Branch: >FF not allowed
3:  LABEL 3
    DPY-1ST SUSPECT TYPE <0-F,ENTER         Ask for 1st suspect
    DPY-+>_
    EXECUTE PROGRAM 27                      Display suspect type
    EXECUTE PROGRAM #4                      Wait till key pressed
    REGE = REGA                             Save suspect type in reg E
    REGA = 0                                Set global: clear type parameter
    IF REGB = 1C GOTO 4                     Branch: ENTER pressed
    REGA = REGB                             Set global: save device type
    IF F >= REGB GOTO 3                     Branch: 1-F types allowed
    REGA = REGE                             Illegal type: restore last good one
    DPY-+#                                  Beep for erroneous entry
    GOTO 3                                  Branch: loop till ENTER pressed
4:  LABEL 4
    REG3 = 0                                Clear 1st suspect #
    IF REGE = 0 GOTO 7                      Branch: 1st suspect type = 0
    DPY-1ST SUSPECT NUMBER <256 =           Ask for suspect #
    DPY-+ \3                                Save in reg 3
    IF REG3 > FF GOTO 4                     Branch: >FF not allowed
    REG3 = REGE SHL SHL SHL SHL SHL SHL     Merge suspect type with # in reg 3
SHL SHL OR REG3
5:  LABEL 5
    IF REG3 = 0 GOTO 7                      Branch: no 1st suspect
    DPY-2ND SUSPECT TYPE <0-F,ENTER         Ask for 2nd suspect type
    DPY-+>_
    EXECUTE PROGRAM 27                      Display 2nd suspect type
    EXECUTE PROGRAM 4                       Wait for operator to press key
    REGF = REGA                             Save suspect type in reg F
    REGA = 0                                Set global: clear type parameter
    IF REGB = 1C GOTO 6                     Branch: ENTER pressed
    REGA = REGB                             Set global: save device type
    IF F >= REGB GOTO 5                     Branch: 1-F types allowed
    REGA = REGF                             Illegal type: restore last good one
    DPY-+#                                  Beep for erroneous entry
    GOTO 5                                  Branch: loop till ENTER pressed
6:  LABEL 6
    REG2 = 0                                Clear 2nd suspect #
    IF REGF = 0 GOTO 7                      Branch: no 2nd suspect type
    DPY-2ND SUSPECT NUMBER <256 =           Ask for 2nd suspect #
    DPY-+ \2                                Save in reg 2
    IF REG2 > FF GOTO 6                     Branch: >FF not allowed
    REG2 = REGF SHL SHL SHL SHL SHL SHL     Merge suspect type with # in reg 2
SHL SHL OR REG2
7:  LABEL 7
    DPY-PRESS 0=SIG, 1=LEVEL,               Ask for sig/level/count mode
    DPY-+ 2=COUNT
    EXECUTE PROGRAM 4                       Wait for operator to press key
    REG5 = REGB                             Save key in reg 5
    IF REG5 > 2 GOTO 7                      Branch: >2 not allowed
8:  LABEL 8
    DPY-PRESS 0=FREE 1=ADRS 2=DATA          Ask for sync mode
    DPY-+ SYNC
    EXECUTE PROGRAM 4                       Wait for operator to press key
    REG4 = REGB                             Save key in reg 4
    IF REG4 > 2 GOTO 8                      Branch: >2 not allowed
    IF REG4 > 0 GOTO 9                      Branch: A or D sync selected
    IF REG5 > 0 GOTO 9                      Branch: not sig and free-run
```

```
   DPY-#NO FREE-RUN SIGNATURES                  Display error message
   REGF = 40                                    Set delay parameter
   EXECUTE PROGRAM 9                            Delay to see display
   GOTO 7                                       Loop for re-entry of mode & sync
9: LABEL 9
   REG1 = 0                                     Clear setup program number
   REGA = 0                                     Clear stimulus program number
   DPY-SETUP PGM= \A; STIMULUS PGM              Ask for setup/stimulus pgm #s (dec)
   DPY-+ =\1                                    Save in reg A and 1
   IF REG1 > 63 GOTO 9                          Branch: >99 decimal not allowed
   IF REGA > 63 GOTO 9                          Branch: >99 decimal not allowed
   REG1 = REGA SHL SHL SHL SHL SHL SHL          Merge pgm #s together in reg 1
SHL SHL OR REG1
   REG8 = REG7 SHL SHL SHL SHL SHL SHL          Merge probe chip & pin # to reg 8
SHL SHL OR REG6
   REG9 = REG2 SHL SHL SHL SHL SHL SHL          Merge 1st & 2nd suspects to reg 9
SHL SHL SHL SHL SHL SHL OR REG3
   REG9 = REG9 SHL SHL SHL SHL OR REG0          Merge probe device type to reg 9
   REG9 = REG9 SHL SHL OR REG4                  Merge sync type to reg 9
   REG9 = REG9 SHL SHL OR REG5                  Merge sig/level/count type to reg 9
   EXECUTE PROGRAM 3                            Enable key interrupt
   REG4 = 7F                                    Set min count to max
   REG5 = 0                                     Set max count to min
   IF REGA = 0 GOTO A                           Branch: no setup program
   EXECUTE PROGRAM REGA                         Execute setup program
A: LABEL A                                      Loop to get known good result
   IF REGB = 1D GOTO 7                          Branch: restart if CLEAR pressed
   IF 40 > REGB GOTO C                          Branch: loop if no key pressed
   DPY-____                                     Clear display
   REGA = REG0                                  Set global: device type to probe
   EXECUTE PROGRAM 27                           Display type to probe
   DPY-+@7-@6                                   Display device number and pin
   REGA = FFFFF                                 Set global: no expected result
   EXECUTE PROGRAM 26                           Display sig/level/count type
   REGC = REG1                                  Set global: setup/stimulus pgm #s
   EXECUTE PROGRAM 23                           Stimulate, take, display reading
   DPY-+ CONT                                   Prompt to press CONT when done
   REGF = 50                                    Set delay parameter
   EXECUTE PROGRAM 9                            Delay to see reading
   IF 2 > REG9 AND 3 GOTO A                     Branch: not event count
   IF REG5 > REGA GOTO B                        Branch: old max > new count
   REG5 = REGA                                  Save new max count
B: LABEL B                                      Old max > new count
   IF REGA > REG4 GOTO A                        Branch: new count > min count
   REG4 = REGA                                  Save new min count
   GOTO A                                       Branch: do next reading
C: LABEL C                                      Routine to enter known-good result
   IF REG9 AND 3 = 2 GOTO E                     Branch: event count mode
   REGD = REGA                                  Set global: save reading
   REG7 = 0                                     Clear min count
   REG6 = REGD AND FFFF                         Get signature
   IF REG9 AND 3 = 0 GOTO D                     Branch: signature mode
   REG6 = REGD SHR SHR SHR SHR SHR SHR          Get logic level history
SHR SHR SHR SHR SHR SHR SHR SHR SHR SHR
SHR SHR SHR SHR SHR SHR SHR SHR AND 7
   DPY-ENTER LVL (4,2,1=L,X,H) $6               Ask for history bit pattern
   DPY-+ = /6                                   Save in reg 6
   IF REG6 > 7 GOTO C                           Branch: >7 not allowed
   REG6 = REG6 SHL SHL SHL SHL SHL SHL          Move history for later merging
SHL SHL
    GOTO F
D: LABEL D                                      Routine to enter known good history
   DPY-ENTER SIGNATURE $6 = /6                  Ask for good sig; save in reg 6
   IF REG6 > FFFF GOTO D                        Branch: >FFFF not allowed
   GOTO F                                       Exit
```

FLUKE®

```
E: LABEL E                                  Routine to enter min and max count
   REG6 = REG4                              Get min count read
   REG7 = REG5                              Get max count read
   DPY-ENTER COUNT MIN @6=\6                Ask for min count; save in reg 6
   DPY-, MAX @7=\7                          Ask for max count; save in reg 7
   IF REG6 > 7F GOTO E                      Branch: >7F min not allowed
   IF REG7 > 7F GOTO E                      Branch: >7F max not allowed
   REG6 = REG6 SHL SHL SHL SHL SHL SHL      Move counts for later merging
SHL SHL
F: LABEL F                                  Display parameters and loop
   REG8 = REG6 SHL SHL SHL SHL SHL SHL      Merge sig, hist, max cnt into reg 8
SHL SHL SHL SHL SHL SHL SHL SHL SHL SHL
OR REG8
   REG8 = REG7 SHL SHL SHL SHL SHL SHL      Merge known-good min count to reg 8
SHL SHL SHL SHL SHL SHL SHL SHL SHL SHL
OR REG8
   REGC = REG1                              Get   setup/stimulus   program   #s
   DPY-REG C=$C; 8=$8; 9=$9                 Display parameters for program 20
   AUX-REG C=$C;  8=$8;  9=$9               Send parameters to RS-232 I/F
   EXECUTE PROGRAM 4                        Wait for operator to press a key
   REGA = 0                                 Set global: initialize device type
   SYNC FREE-RUN                            Reset sync mode to free-run
   GOTO 0                                   Loop to beginning of program


PROGRAM 23   READER   223 BYTES             Stimulate circuit and take readings

Inputs:     Registers C, 8, and 9 as setup by program 20 or 22
Called by:  programs 21 and 22
Calls to:   programs 9, 25, and Stimulus program specified by register C
Outputs:    Register A = actual result

   REG1 = REGD                              Save any D register value
   IF REG9 AND C = C GOTO D                 Branch: invalid sync parameter
   IF REG9 AND 3 = 3 GOTO D                 Branch: invalid mode parameter
   REG2 = REGC AND FF                       Get stimulus program number
0: LABEL 0
   SYNC DATA                                Initialize with Data sync mode
   IF REG9 AND 8 > 0 GOTO 1                 Branch: Data sync flag is set
   SYNC ADDRESS                             Enable Address sync mode
   IF REG9 AND 4 > 0 GOTO 1                 Branch: Address sync flag is set
   IF REG9 AND 3 = 0 GOTO 1                 Branch: force adrs sync if sig mode
   SYNC FREE-RUN                            No other sync: select free-run sync
1: LABEL 1
   READ PROBE                               Initialize signature/count/history
   READ PROBE                               Take quick reading
   IF REG2 = 0 GOTO 2                       Branch: no stimulus program #
   EXECUTE PROGRAM REG2                     Stimulate circuit under test
   READ PROBE                               Take signature,count, or history
2: LABEL 2
   IF REG9 AND 2 > 0 GOTO 3                 Branch: event counts selected
   IF REG9 AND 1 > 0 GOTO 4                 Branch: logic history selected
   REGA = REG0 SHR SHR SHR SHR SHR SHR      Mask out history and count
SHR SHR AND FFFF
   DPY-+$A                                  Display signature taken
   GOTO E                                   Exit
3: LABEL 3                                  Event counts:
   REGA = REG0 AND 7F                       Mask out signature and history
   DPY-+@A                                  Display counts in decimal
   GOTO E                                   Exit
4: LABEL 4                                  Logic level history
   REGA = REG0 AND 7000000                  Mask out signature and count
   EXECUTE PROGRAM 25                       Display history
   GOTO E                                   Exit
D: LABEL D                                  Stop for invalid reg 9 parameters
```

FLUKE®

```
   DPY-+#BAD REG9=$9                         Display error message
   STOP                                      Stop the program
   GOTO F                                    Exit on continue
E: LABEL E                                   Exit
   DPY-+,__                                  Add comma and space to display
F: LABEL F                                   Exit
   REGD = REG1                               Restore value to reg D
```

PROGRAM 24   MONITOR   236 BYTES   Ensure probe in or out of circuit

```
Inputs:      Reg A = 0 to insert probe, 1 to remove probe; reg 8 and 9 as
             specified by program 20.
Called by:   program 21
Calls to:    program 3
Outputs:     Reg B = 40 if no key pressed; 41 if CONT during loop; key value
```

```
   SYNC FREE-RUN                             Set free-run to enable async probe
   IF REGA > 0 GOTO 0                        Branch: remove probe
   IF REG9 AND 1 = 0 GOTO 0                  Branch: no CONT if not history
   IF REG8 AND 7000000 = 2000000 GOTO 4      Branch: not seeking invalid state
0: LABEL 0                                   Loop point for repeating check
   REG1 = 10                                 Initialize pass counter
1: LABEL 1
   READ PROBE                                Take probe reading
   IF REGA = 0 GOTO 2                        Branch: insert probe
   IF REG0 AND 5000000 = 0 GOTO 3            Branch: high/low received
   DPY-REMOVE PROBE                          Display message
   GOTO 0                                    Loop till probe is removed 10 tries
2: LABEL 2                                   Wait till probe is inserted
   IF 40 > REGB GOTO D                       Branch: operator pressed key
   IF REG0 AND 5000000 = 0 GOTO 0            Branch: high/low not received
3: LABEL 3                                   High/low received:
   DEC REG1                                  Decrement pass counter
   IF 40 > REGB GOTO F                       Branch: operator pressed key
   IF REG1 > 0 GOTO 1                        Branch: loop till 10 good passes
   GOTO F                                    Exit after 10 good passes
4: LABEL 4                                   Entry if needed to press CONT
   DPY-+, CONTINUE                           Display message
5: LABEL 5
   IF REGB = 40 GOTO 5                       Wait till operator presses key
   IF REGB = 25 GOTO 6                       Branch: operator pressed CONT
   GOTO F                                    Exit
6: LABEL 6
   EXECUTE PROGRAM 3                         Reenable key interrupt
   GOTO F                                    Exit
D: LABEL D                                   Key pressed during check routine
   IF REGB = 25 GOTO E                       Branch: key was CONT
   GOTO F                                    Exit: key was not CONT
E: LABEL E                                   CONT key was pressed
   EXECUTE PROGRAM 3                         Re-enable key interrupt
   REGB = 41                                 Set global: CONT pressed
   DPY-+                                     Add space to display
F: LABEL F                                   Exit
```

PROGRAM 25   D-HISTORY   103 BYTES   Display logic history

```
Inputs:      Register A = history Y000000, where Y bits are LXH, or FFFFF (from)
             program 22 for no history.
Called by:   programs 23 and 26
Calls to:    none
Outputs:     display only
```

```
   IF REGA AND 7000000 > 0 GOTO 0            Branch: some history indicated
   DPY-+NONE                                 Display message of no history
   GOTO 3                                    Branch: exit
0: LABEL 0                                   Display type of history
   IF REGA AND 1000000 = 0 GOTO 1            Branch: not HIGH
   DPY-+H                                    display H
1: LABEL 1
   IF REGA AND 2000000 = 0 GOTO 2            Branch: not INVALID STATE
   DPY-+X                                    Display X
2: LABEL 2
   IF REGA AND 4000000 = 0 GOTO 3            Branch: not LOW
   DPY-+L                                    Display L
3: LABEL 3
```

PROGRAM 26   D-EXPECTED   169 BYTES   Display mode and expected result

```
Inputs:      Reg A = expected result from program 21, FFFFF from program 22;
             Reg 9 = setup by program 20, 22
Called by:   programs 21, 22
Calls to:    program 25
Outputs:     Reg A = expected sig, history, min count; reg D = exp max count
```

```
   IF REG9 AND 1 = 1 GOTO 2                  Branch: logic history selected
   IF REG9 AND 2 = 2 GOTO 3                  Branch: event count selected
1: LABEL 1                                   Signature selected
   DPY-+SIG                                  Display reading type
   IF REGA > FFFF GOTO 5                     Branch: no expected result
   DPY-+ $A                                  Display expected signature
   GOTO 5                                    Exit
2: LABEL 2                                   Logic history selected
   DPY-+LEVEL ____                           Display reading type
   IF REGA > FFFF GOTO 5                     Branch: no expected result
   REGA = REG8 AND 7000000                   Mask out all but history
   EXECUTE PROGRAM 25                        Display history
   GOTO 5                                    Exit
3: LABEL 3                                   Event count selected
   IF REGA > FFFF GOTO 4                     Branch: no expected result
   REGD = REGA AND 7F                        Mask out all but maximum count
   REGA = REGA SHR SHR SHR SHR SHR SHR       Mask out all but minimum count
SHR SHR AND 7F
   DPY-+CNT [redacted]@A—@D                  Display minimum and maximum count
4: LABEL 4                                   Display for program 22
   DPY-+COUNT                                Display reading type
5: LABEL 5                                   Exit
   DPY-+ =_                                  Add equal sign to display
```

PROGRAM 27   D-DEVICE   284 BYTES   Displays device type

```
Inputs:      Reg A = value 0 through F to display a device (0=none)
Called by:   programs 21 and 22
Calls to:    none
Outputs:     display only
```

```
   REG1 = REGA AND F                         Mask out all but the lower nibble
   IF REG1 = 0 GOTO F                        Branch: to selected display code
   IF REG1 = 1 GOTO 1
   IF REG1 = 2 GOTO 2                        Note: modify display steps to
   IF REG1 = 3 GOTO 3                        tailor this program to your needs
   IF REG1 = 4 GOTO 4
```

```
    IF REG1 = 5 GOTO 5
    IF REG1 = 6 GOTO 6
    IF REG1 = 7 GOTO 7
    IF REG1 = 8 GOTO 8
    IF REG1 = 9 GOTO 9
    IF REG1 = A GOTO A
    IF REG1 = B GOTO B
    IF REG1 = C GOTO C
    IF REG1 = D GOTO D
    IF REG1 = E GOTO E
    DPY-+                    Display item F (Available for use)
    GOTO F                   Exit
1:  LABEL 1
    DPY-+U                   Display item 1: Integrated circuit
    GOTO F
2:  LABEL 2
    DPY-+Q                   Transistor
    GOTO F
3:  LABEL 3
    DPY-+R                   Resistor
    GOTO F
4:  LABEL 4
    DPY-+C                   Capacitor
    GOTO F
5:  LABEL 5
    DPY-+CR                  Diode
    GOTO F
6:  LABEL 6
    DPY-+SW                  Switch
    GOTO F
7:  LABEL 7
    DPY-+LED                 Light emitting diode
    GOTO F
8:  LABEL 8
    DPY-+KEY                 Pushbutton or key
    GOTO F
9:  LABEL 9
    DPY-+K                   Relay
    GOTO F
A:  LABEL A
    DPY-+P                   Plug
    GOTO F
B:  LABEL B
    DPY-+J                   Jack
    GOTO F
C:  LABEL C
    DPY-+X                   IC socket
    GOTO F
D:  LABEL D
    DPY-+BP                  Backplane
    GOTO F
E:  LABEL E
    DPY-+                    Available for your use
    GOTO F
F:  LABEL F
```

John Fluke Mfg. Co., Inc.
P.O. Box C9090, Everett, WA 98206
800-426-0361 (toll free) in most of U.S.A.
206-356-5400 from AK, HI, WA
206-356-5500 from other countries

Fluke (Holland) B.V.
P.O. Box 5053, 5004 EB, Tilburg, The Netherlands
Tel. (013) 673973, TELEX 52237
Phone or write for the name of your local Fluke representative.

Printed in U.S.A.

B0138A-01U8108/SE EN