33D7-32-12-11

27 SEP 1989

9100 Series

Applications Manual

P/N 813840

FLUKE®

1 FEBRUARY 1988

LIMITED WARRANTY

John Fluke Mfg. Co., Inc. (Fluke) warrants your 9100/9105A to be free from defects in material and workmanship under normal use and service for 90 days from the date of shipment. Software and firmware products are provided "AS IS." We do not warrant that software or firmware products will be error free, operated without interruption or that all errors will be corrected. This warranty extends to you if you are the original purchaser and does not apply to fuses, batteries or any product which, in our sole opinion, has been subject to misuse, alteration or abnormal conditions of operation or handling.

To obtain warranty service, contact a Fluke Service Center or send the product, with the description of the difficulty, postage prepaid, to the nearest Fluke Service Center. Fluke assumes no risk for damage in transit.

Fluke will, at our option, repair or replace the defective product free of charge. However, if we determine that the failure was caused by misuse, alteration, or abnormal condition of operation or handling, you will be billed for the repair. The repaired product will be returned to you, transportation prepaid.

THIS WARRANTY IS EXCLUSIVE AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR USE. FLUKE WILL NOT BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OR LOSS WHETHER IN CONTRACT, TORT, OR OTHERWISE.

APAGED
30 APR 1990 TLS

Contents

(This page is intentionally blank.)

Figures

(This page is intentionally blank.)

Where Am I?

Getting Started		A description of the parts of the 9100A/9105A, what they do, how to connect them, and how to power up.
Automated Operations Manual		How to run pre-programmed test or troubleshooting procedures.
Technical User's Manual		How to use the 9100A/9105A keypad to test and troubleshoot your Unit Under Test (UUT).
Applications Manual	You Are Here	How to design test or troubleshooting procedures for your Unit Under Test (UUT).
Programmer's Manual		How to use the programming station with the 9100A to create automated test or troubleshooting procedures.
TL/1 Reference Manual		A description of all TL/1 commands arranged in alphabetical order for quick reference.

(This page is intentionally blank.)

Section 1
Introduction

ORGANIZATION OF THIS MANUAL 1.1.

This manual provides an organized approach to testing and troubleshooting a UUT (Unit Under Test) with the 9100A/9105A. The intended reader is someone who will be writing test programs or test procedures for use with the 9100A/9105A.

Additional information on the various parts of the 9100A/9105A system is available in the *Getting Started* booklet. More information about using the operator's keypad for testing and troubleshooting is available in the *Technical Users's Manual.* And more information on programming the 9100A/9105A is available in the *Programmer's Manual* and in the *TL/1 Reference Manual.*

This manual is organized into three major parts:

Sections 1 to 3 give an overview of the capabilities of the 9100A/9105A and the process of developing functional tests and automated troubleshooting procedures.

Section 4 describes some typical functional blocks for a microprocessor-based UUT. For each typical functional block,

you will find a summary of things to consider for testing and troubleshooting, a procedure for using the operator's keypad of the 9100A/9105A for functional testing, a 9100A/9105A programmed functional test, and a set of stimulus programs.

NOTE

Each of the functional blocks described in Section 4 are parts of a real UUT, the Fluke Demo/Trainer. It is not necessary that you have the Demo/Trainer UUT to use this manual, but you may wish to purchase the Demo/Trainer from Fluke so you can try out the example procedures and programs.

Sections 5 to 7 show you how to build on the block functional tests to develop functional tests for your whole UUT and how to develop automated troubleshooting procedures using Guided Fault Isolation (GFI).

PREPARING FOR TESTING AND TROUBLESHOOTING 1.2.

The 9100A/9105A is both a testing and a troubleshooting system. As a test station, it determines whether functional blocks of digital circuitry pass or fail. As a troubleshooting station, it determines which nodes or circuit connections are faulty.

The 9100A/9105A has many built-in functions which are useful for functional testing, stimulation of nodes, and measurement of node or component behavior. In addition, the 9100A has a powerful programming language, called TL/1, that is used to customize the capabilities of the 9100A/9105A to match the testing and troubleshooting requirements for your UUT.

The Programmer's Interface option of the 9100A is used to enter UUT information and to create programs that become the building blocks for automated testing and troubleshooting. This interface also provides an automated process for collecting and storing node responses from a known-good UUT. When the 9100A/9105A is used for testing and troubleshooting,

measurements on a node are compared with these stored, known-good node responses to determine whether the measured node response is good or bad.

The 9100A is easily programmed. The operator's keypad and display allow you to explore the operation of your UUT by pressing keys on the keypad. Then, as you develop successful test and troubleshooting procedures, you can put these procedures into TL/1 programs to automate the process. Or, if you prefer, you can write the TL/1 programs directly and then check their operation with the debugger built into the 9100A.

The 9100A/9105A is very flexible; it can be used with several different levels of investment in programming. As you increase the level of programming, you increase the degree of automation and the ease of testing and troubleshooting. Five typical levels of programming effort are summarized below and are also shown graphically in Figure 1-1.

- *No programming effort:* Use the keys of the operator's keypad to initiate testing and troubleshooting actions. This level is appropriate for testing or troubleshooting one-of-a-kind UUTs, where investment in programmed testing and troubleshooting is not cost-effective. It is also valuable for keystroke testing and troubleshooting prior to the completion of programmed testing and troubleshooting. Keystroke testing and troubleshooting requires a skilled technician operator.
- *Level 1 Programming:* Create stimulus programs that cause predictable activity at a node and characterize that node activity on a known-good UUT. You may choose to create the node list and the reference designator list at this level also. If you do so, you will be able to backtrace from a bad node to the fault which causes it. You do this by pressing the GFI key on the operator's keypad and specifying the failing node as the starting point.
- *Level 2 Programming:* Create functional tests for each functional block of your UUT. These tests determine whether the functional block passes or fails. Some block

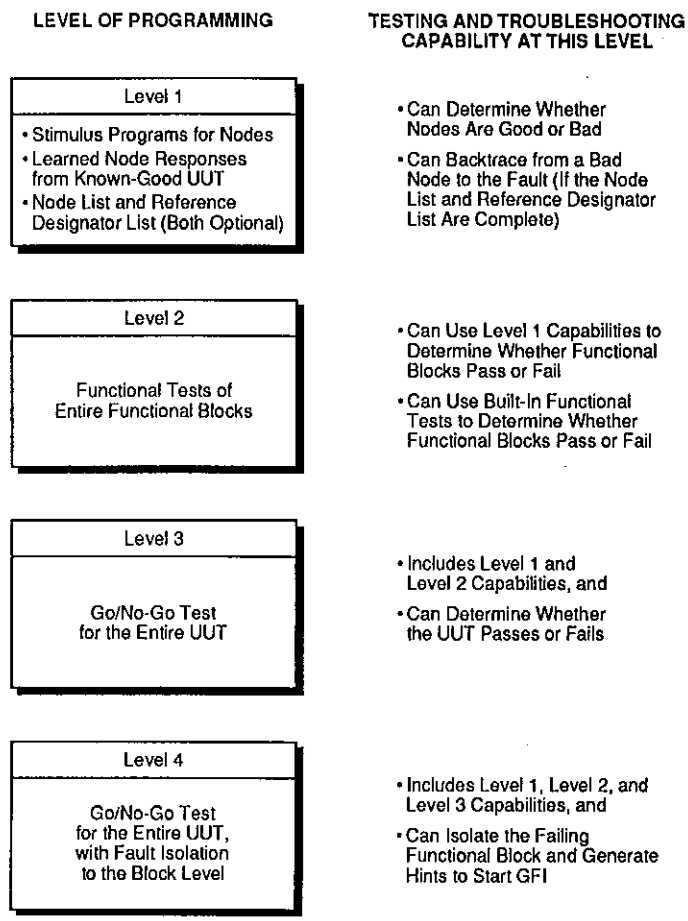


Figure 1-1: Recommended Programming Sequence

functional tests will use stimulus programs from Level 1, and others will have independent functional test programs.

- *Level 3 Programming:* Create a go/no-go test for the entire UUT, by using all of the necessary functional block tests to create a functional test of the whole UUT. This test determines whether a UUT is good or bad, but does not usually isolate the fault.
- *Level 4 Programming:* Add procedures to the go/no-go test that will isolate the faulty block for any UUTs that fail the go/no-go functional test. This addition to the go/no-go test provides efficient starting points for automated troubleshooting with GFI. If you have not already done so in Level 1, create the node list and the reference designator list. Your program will then be able to backtrace from a bad node to the fault which causes it. Or you can backtrace by pressing the GFI key on the operator's keypad and specifying a failing node as the starting point.

The 9100A/9105A is the center of a expandable system. For example, fixturing can be added to improve functional test throughput in high-volume applications. In addition, the 9100A/9105A can be integrated with manufacturing systems or host computers.

WHERE TO BEGIN 1.3.

The 9100A/9105A system can be operated manually from the operator's keypad in an "immediate" (keystroke) mode, or it can be programmed in TL/1 with functional tests and GFI procedures using the programmer's interface of the 9100A.

A good overview of the full capabilities of the 9100A/9105A will be helpful before you begin using it in either mode. One good way to explore the use of the 9100A/9105A is to adopt the techniques shown in this manual to your own UUT. While reading Section 4, you might try some of the reads, writes, and built-in tests on your own UUT. To try Guided Fault Isolation (GFI), you could treat a small portion of your UUT as if it were the entire system to be tested and diagnosed. Two or three

components connected to the microprocessor bus are usually sufficient for such an introductory exploration.

This manual does not assume that you know the TL/1 programming language, although examples of TL/1 programs are included throughout the manual. As you look over these programs and their explanations, you will find many of them quite understandable. However, in some places, you may want to refer to the *Technical User's Manual,* the *Programmer's Manual,* or the *TL/1 Reference Manual* to learn how specific keys or commands work.

Section 2
Overview of Testing and Troubleshooting

"Testing" determines whether a circuit is good (passes) or bad (fails). "Troubleshooting" finds the faulty component or node causing a circuit to fail.

Before microprocessors, a circuit board was tested by applying a sequence of patterns to inputs at the board's edges or at selected nodes within the board's circuitry and then measuring the output. However, for circuit boards that use microprocessors, the most comprehensive coverage is provided by controlling the UUT from the microprocessor bus. One common method of doing this is to plug in a tester at the microprocessor socket.

Testers that control the microprocessor bus must be able to apply stimuli and capture responses at specific times during the cycles. As an example, consider a buffer on a microprocessor data bus: since data is only stable during a small period of the bus cycle, the outputs of the buffer must be measured at the proper time during the bus read/write cycle.

The basic functions of a test system and the basic functions of a troubleshooting system are similar. During either task, the system must emulate bus cycles and measure levels and signal patterns. But the two tasks have different goals. During testing, the goal is to determine whether a UUT is good or bad; it is not necessary to know where the faults are. However, in

troubleshooting the goal is to determine what component is bad or what node is bad so that the UUT can be repaired.

Figure 2-1 shows a testing, troubleshooting, and repair cycle. Some users consider testing and troubleshooting to be completely separate tasks. Other users consider them to be almost identical. In situations where volumes of each type of board tested are high, and where many of the boards are likely to be good, the testing and troubleshooting tasks are often separated. But if board volumes are low or if many of the boards tested are faulty, the testing and troubleshooting tasks are often combined into a single process.

The 9100A/9105A can perform testing and troubleshooting as a single task or as separate tasks. In either case, the system's TL/1 programs are very similar because of the modular structure encouraged by the 9100A programming environment. This manual discusses a broad variety of test and troubleshooting techniques; you can then determine how the techniques should be linked and to what degree the entire process should be automated for your application.

EMULATIVE TESTING 2.1.

The 9100A/9105A is an emulative tester and troubleshooter. By taking control of the UUT's microprocessor bus, the 9100A/9105A can perform all operations, apply all stimuli, and capture any responses that the UUT microprocessor could.

The 9100A/9105A is designed for testing microprocessor-based hardware. The emulative testing approach of the 9100A/9105A should not be confused with in-circuit emulators which also plug into the microprocessor socket and are designed to test software. The in-circuit emulators are difficult to use for board testing because they work with assembly language (which is different from one microprocessor to another). They also require the use of breakpoints to allow examination of UUT registers and memory to check out operation of the UUT. In contrast, the TL/1 programming language of the 9100A/9105A has

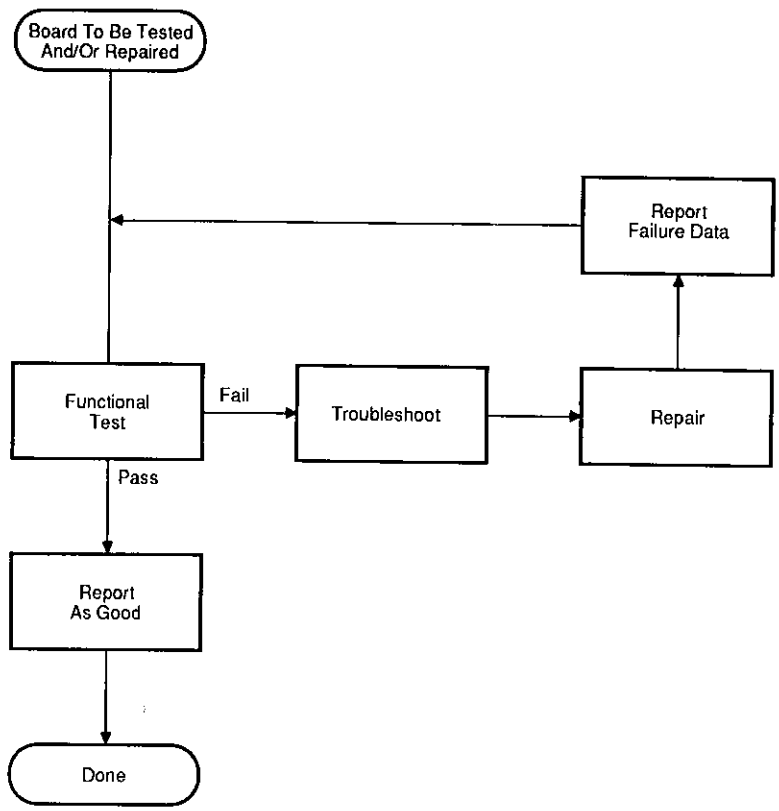


Figure 2-1: Testing, Troubleshooting, and Repair

commands to perform read or write accesses without requiring that you write any assembly language.

The basic elements of the 9100A/9105A system's emulative testing are:

- Stimulation and response sensing at the microprocessor bus by the pod.
- Stimulation of circuitry by the pod, probe, and I/O module.
- Measurement of stimulation responses with the pod, probe, and I/O module as the signals propagate throughout the UUT.
- High-level programming language (independent of the target microprocessor) to control microprocessor accesses and operations.

Figure 2-2 illustrates these capabilities. The method of emulative testing allows the pod to read from and write to any components that the microprocessor can access. The pod can initialize and program components in the UUT, such as DMA controllers, PIAs, serial ports, and video controllers.

In addition to controlling the UUT from the microprocessor bus, the pod senses loaded or faulty lines at the socket where the pod plugs into the UUT. For example, if a data line has a short to ground, the pod will detect that the line cannot be driven when the pod attempts to drive the line high.

The I/O modules and the probe can measure and stimulate all of the UUT's digital circuitry, including circuits not directly accessible by the pod. The pod, I/O module, and probe are used together or individually to provide a stimulus and to capture responses.

The 9100A/9105A can characterize nodes with CRC signatures, level histories (asynchronous or synchronous), transition counts, and frequencies using the single-point probe or 40-line I/O modules. I/O modules accommodate clip modules that fit

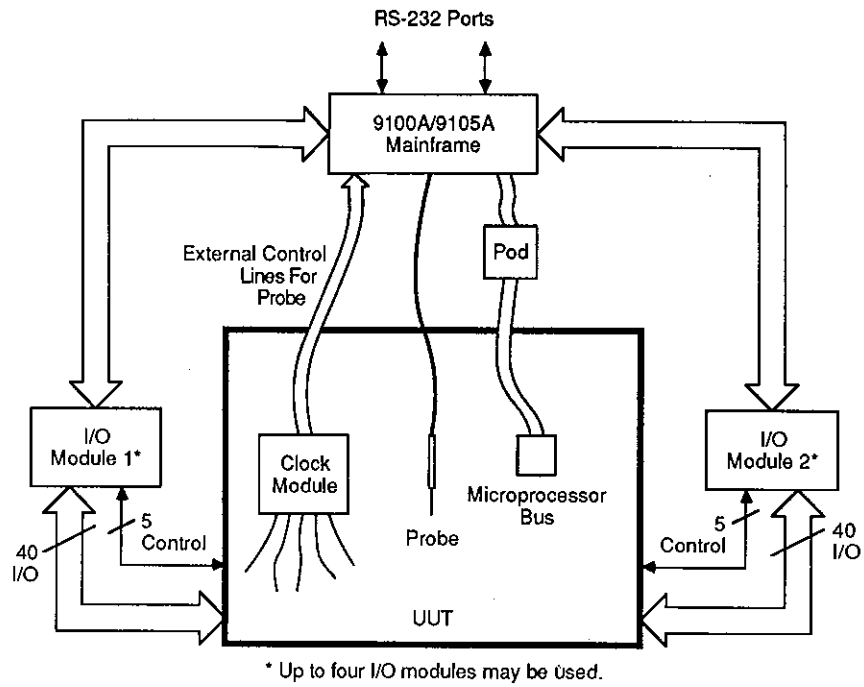


Figure 2-2: Emulative Testing With the 9100A/9105A

various IC packages. The I/O modules can also be used in fixturing.

When the 9100A/9105A stimulates the UUT through the microprocessor bus, an I/O module or the probe can measure the signals as they propagate through the UUT. Or, the I/O modules can stimulate nodes and the pod can measure the activity from the microprocessor bus.

A powerful feature of the 9100A/9105A is that it can perform measurements which are synchronized to microprocessor operations. For example, consider the microprocessor bus. It is a flurry of activity when examined with an oscilloscope, but the 9100A/9105A can control this activity and can examine the signals on the data bus at times when the signals are valid.

The probe and I/O modules can be synchronized to data, address, and other pod cycles, as well as to external Clock, Start, Stop and Enable inputs provided on the 9100A/9105A's I/O module and clock module. The external sync modes are valuable for measuring events asynchronous to the microprocessor, such as video signals and free-running counters.

NODE CHARACTERIZATION 2.2.

Node characterization is the process of finding a description of the correct activity at a node, given an appropriate stimulus to the UUT to exercise the node. A quality characterization is one that is repeatable from one measurement to another, from one UUT to another, and from one day to another. In addition, incorrect activity at the node should result in a value that is different from the characterization for correct node activity. The 9100A/9105A uses the probe or the I/O module to measure five node characteristics:

- *CRC signature:* This measures high and low levels relative to a series of events (called "clock" or "sync") and then encodes a Cyclic Redundancy Check (CRC) number representing both level and timing. The signature, if

stable, is the most accurate characterization of a node. If the node changes states at or near the clock transition, the signature is considered marginal because a slight relative time change between clock and data will change the signature.

- *Asynchronous level history:* This indicates whether the node was ever at a high, low, or invalid level at any time during a specified period.
- *Clocked (synchronous) level history:* This indicates whether the node was ever at a high, low, or invalid level at any clock or sync edge during a specified period.
- *Transition count:* This measures how many times the node goes low-to-high during the measurement period. When a given node is measured, a single count value is returned. Learned responses stored in a response file, however, may appear as a range of counts. If a range of counts is specified, the measurement will be considered good if it is within the specified range.
- *Frequency:* This measurement is done during a set time interval and is unrelated to clock or sync modes. As with transition counts, learned responses stored in a response file may appear as a range of frequencies.

STIMULUS AND MEASUREMENT CAPABILITIES 2.3.

Figure 2-3 is an overview of the stimulus and measurement capabilities of the 9100A/9105A. The devices used for this include the:

- Pod.
- Probe (with clock module).
- I/O module.

The following sections describe the capabilities of each of these devices.

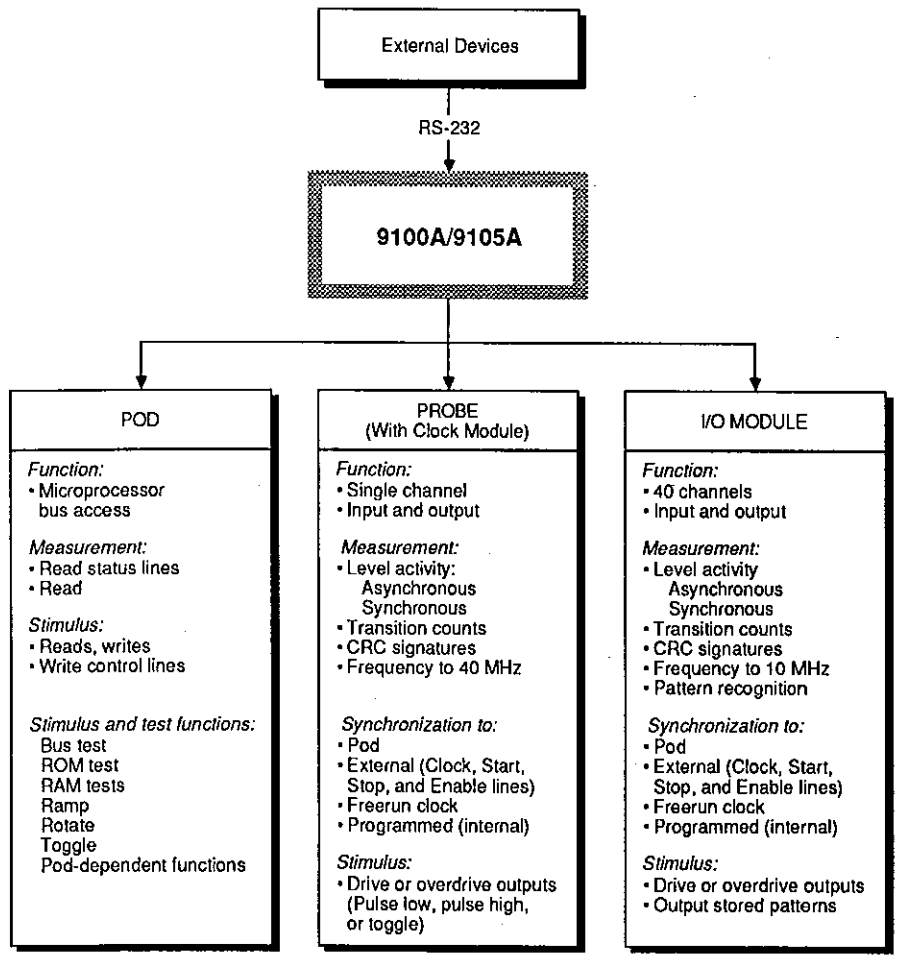


Figure 2-3: 9100A/9105A Stimulus and Measurement Capability

Pod Capabilities 2.3.1.

The Fluke interface pods provide the interface between the 9100A/9105A and the microprocessor bus of a UUT. The pod has two modes of operation: normal mode (where the microprocessor in the pod exercises the UUT microprocessor bus while monitoring the activity on this bus) and RUN UUT mode (where the microprocessor in the pod runs programs stored in UUT memory). A wide variety of stimulus and measurement commands are available either from the operator's keypad or from programs written for automated implementation.

Additional information about pods, their use, and their specifications is contained in section 2.4 of the *Technical User's Manual*, the pod manual for the pod you are using, the *Supplemental Pod Information for 9100A/9105A Users Manual*, and section 3.5 of the *Programmer's Manual.*

Probe Capabilities (With The Clock Module) 2.3.2.

The probe can provide either measurement or output at any selected node of a UUT.

The probe can measure CRC signatures, asynchronous level histories, clocked (synchronous) level histories, transition counts, and frequencies. It has built-in lights to show the current asynchronous level (or levels) at the probe tip or to show the level (or levels) last seen by the synchronous level history latches. The probe can be set up to use one of three different sets of logic thresholds for its measurements: TTL, CMOS, or RS-232.

The probe can also be used as an output device to output a series of pulses. The pulses can be high, low, or can toggle between high and low. The probe has sufficient drive capability (200mA for less than 10µsec or 5mA continuously) to overdrive most circuit nodes.

The probe is synchronized to other events by four synchronization modes: freerun clock, pod data or address

sync, external sync (using the external control lines of the Clock Module), and internal sync (for use under program control only). The external control lines of the Clock Module use TTL-level thresholds.

Additional information about the probe, its use, and its specifications is contained in section 2.5 of the *Technical User's Manual,* Appendix D of the *Technical User's Manual,* and section 3.6 of the *Programmer's Manual.*

I/O Module Capabilities 2.3.3.

Each I/O module can make simultaneous connection with up to 40 UUT nodes. I/O module adapters provide an interface between the general-purpose connectors on the I/O module and components on a UUT. The smaller clip modules can be plugged into either side A or side B of the I/O modules, and the larger clip modules use both connectors.

An I/O module can measure CRC signatures, asynchronous level histories, clocked (synchronous) level histories, transition counts, and frequencies. Unlike the probe, an I/O module can measure multiple pins at the same time. An I/O module can be set up to use one of two different sets of logic thresholds for its measurements: TTL and CMOS.

In addition, I/O modules can recognize words that exist across selected UUT nodes. Recognition of specified words generates a Data Compare Equal (DCE) condition, sends a signal out the DCE pin at the side of the I/O module, and terminates any RUN UUT in progress.

I/O module outputs can be latched high or low, pulsed high or low, or allowed to float (high-impedence). In addition, it can use TL/1 commands to drive patterns out of each output. Responses can be measured at any pin while the I/O module is driving a pattern. An I/O module has sufficient drive capability (2A for less than 10µsec or 200mA continuously) to overdrive most circuit nodes.

An I/O module is synchronized to other events by four synchronization modes: freerun clock, pod data or address sync, external sync (using the external control lines located on the I/O module itself), and internal sync (for use under program control only). The external control lines use TTL-level thresholds.

Additional information about the I/O modules, their use, and their specifications is contained in section 2.5 of the *Technical User's Manual,* Appendix D of the *Technical User's Manual,* and section 3.6 of the *Programmer's Manual.*

TESTING AND TROUBLESHOOTING WITH THE 9100A/9105A

2.4.

The 9100A/9105A can be used for:

- Functional testing.
- Troubleshooting.
- Combined testing and troubleshooting.

As a functional tester, the 9100A/9105A can determine whether a UUT passes or fails a series of tests. As a troubleshooter, the the system can first isolate the failing functional block and then identify a starting location from which detailed fault isolation can locate the node or component causing the failure.

When testing and troubleshooting are performed at the same test station, the 9100A/9105A performs the following sequence of operations:

1. Perform a go/no-go (pass/fail) test of the UUT.
2. Diagnose a failing UUT to determine which functional block is failing.
3. Identify a starting point for fault isolation.
4. Locate the the node or component causing the failure.

If testing and troubleshooting are performed at separate stations, the 9100A/9105A would perform Step 1 at the testing station and Steps 2 through 4 at the troubleshooting station.

In situations where Step 1 is performed by another type of tester, which identifies suspect functional blocks to the 9100A/9105A, the 9100A/9105A can verify that the problem is really in the indicated block before detailed fault isolation is begun. Occasionally, the real problem is in a different functional block than that indicated by functional testing; for example, a functional tester might indicate a fault in the interrupt circuit, whereas the real fault may lie in the serial I/O circuit. If the failure is not in the indicated functional block, the 9100A/9105A at the troubleshooting station can perform its own full functional test to determine the location of the problem.

The 9100A/9105A has very fast built-in functions to test the microprocessor bus, ROM, and RAM, as well as powerful built-in fault condition handling capabilities that ease the communication between the testing functions and the troubleshooting functions.

After stimulus programs and a reference list of parts have been developed for a UUT, the process of testing can be greatly simplified with the TL/1 programming language's *gfi test* command, which uses portions of the 9100A/9105A's Guided Fault Isolation (GFI) database to automate much of the data collection and comparison needed for evaluation of test results.

GFI Troubleshooting:

The 9100A/9105A uses the backtracing method (from bad to good) for its built-in Guided Fault Isolation troubleshooting capability. A functional test locates outputs that appear bad, and GFI starts backtracing from those outputs to locate quickly the failing node. In doing this, GFI uses its database of IC pinouts (the part library, largely supplied by Fluke) and your node list (with part-number references).

The built-in GFI algorithm is efficient at backtracing. However, troubleshooting time can be further reduced by having functional tests provide suggested starting points for GFI (called "GFI hints") as close as possible to the failing node or component. Hints which are close to the fault improve the efficiency of GFI by decreasing the number of nodes that GFI must trace through before reaching the fault.

You can improve GFI's backtracing by:

- Developing functional tests for intermediate functional blocks wherever practical. If a functional test for a major block fails, test the intermediate functional blocks and provide hints which are close to the failure.
- Designing functional tests that, upon failure, measure intermediate nodes in order to provide hints close to the failure. Functional tests can also include fault condition handlers that interpret diagnostic messages to determine where the failure might be located.

(This page is intentionally blank.)

Section 3
Developing Procedures and Programs

UNDERSTANDING THE UUT 3.1.

A UUT should be well understood before functional tests and troubleshooting routines are developed. Taking time at the beginning to study the UUT will result in quicker program development, greater fault coverage, and more accurate fault detection.

Before developing functional test programs and troubleshooting routines:

- Learn what each circuit does, how it works, and how to initialize it.
- Determine the UUT memory map.
- Determine the initialization procedures for each programmable chip.

PARTITIONING THE UUT 3.2.

Circuit partitioning involves dividing the entire circuit into a collection of smaller functional blocks which are easier to

understand and test. It is the first step toward a divide-and-conquer method of testing and troubleshooting and it is time well spent. Once the task is done, the functional blocks can be considered as components, each of which receives inputs and generates outputs. Like an IC, a functional block is suspected of being bad if it has good inputs and bad outputs.

Here are some guidelines for partitioning circuits:

- Group circuits by function, making the functional blocks well-defined pieces of the UUT block diagram and as logically distinct as possible.
- If a functional block is large, subdivide it. This will improve troubleshooting efficiency.
- If failure of a circuit can cause failures to appear in many other parts of the UUT, make that circuit a functional block.
- If a circuit requires a unique test setup, make it a functional block.

An Example of Partitioning (The Demo/Trainer UUT) 3.2.1

The Demo/Trainer UUT (Figure 3-1) is an 80286-based system which includes ROM, Dynamic RAM, Parallel I/O, Video, and Serial I/O circuits. It is available from Fluke as an option and is a good example of 16-bit microcomputer systems. Contact a Fluke representative for information about this option.

The test and troubleshooting examples throughout this manual relate to the Demo/Trainer UUT. With it, you can perform the hands-on tests given in the following sections. The complete UUT nodelist, part-reference list, and schematics are shown in the appendices of the manual.

If you do not have a Demo/Trainer UUT, the examples provide enough information so that you can follow the techniques and sample programs and apply the concepts to your UUT.

1. RS-232 CONNECTOR
2. VIDEO CONNECTOR
3. TEST SWITCHES (S1 THROUGH S4)
4. STATUS LEDs
5. KEYBOARD CONNECTOR
6. RESET BUTTON
7. 80286 MICROPROCESSOR

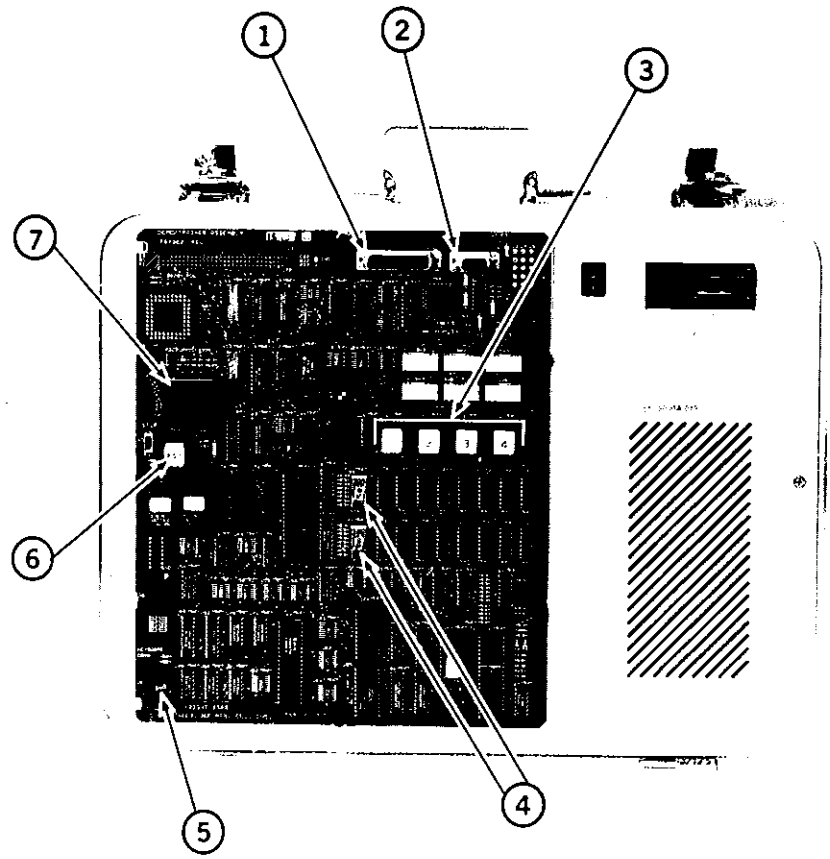


Figure 3-1: Demo/Trainer UUT

A simple block diagram of the Demo/Trainer UUT might show only five blocks: RAM, ROM, Parallel I/O, Serial I/O, and Video. While this is useful as an overview of what the system does, it is inadequate for the development of test and troubleshooting procedures. By subdividing this diagram into smaller sections, we arrive at functional blocks that can be more easily understood. Figure 3-2 shows these smaller blocks, which will be used as examples throughout this manual.

For example, the video circuitry is subdivided into three functional blocks: Video Output, Video Control, and Video RAM. This was done in anticipation that three distinct troubleshooting setups would be needed for the video circuitry. It was also done to reduce troubleshooting time by allowing functional tests to determine which portion of the video circuit has failed before GFI is invoked. Remember, troubleshooting with GFI normally begins at an output node of the failing functional block and backtraces toward good inputs to that block. Subdivision allows GFI to begin backtracing closer to the fault. For similar reasons, the dynamic RAM circuit is subdivided into RAM and Dynamic RAM Timing.

The microprocessor itself is shown in Figure 3-2 as a separate functional block for a good reason: when the pod replaces the microprocessor, it becomes a known-good functional block. All outputs from this circuit can be directly controlled by the 9100A/9105A. The pod checks for drivability on every UUT access and reports if there is a loading problem.

The Bus Buffer is partitioned separately, not for reasons of clarity, but so that it can have its own functional tests. If this circuit has a fault in it, the fault will cause most of the other functional blocks to also fail. So if the UUT fails a functional test, it is more efficient to check the Bus Buffer early in the troubleshooting process.

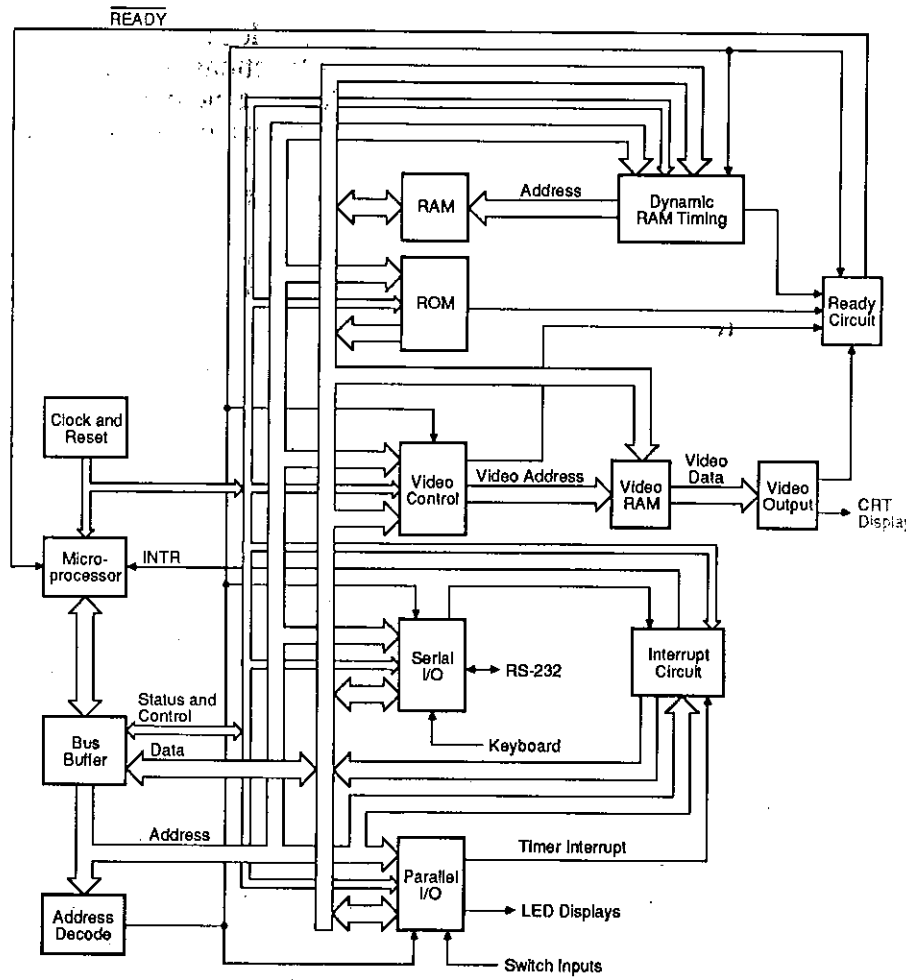


Figure 3-2: Demo/Trainer UUT Functional Blocks

The Advantage of Partitioning 3.2.2

After the partitioning is done, step back and look at the resulting detailed block diagram. Imagine that a functional test has been developed for each individual block. If a novice user has nothing but this block diagram and the collection of individual block tests, he can make a fair degree of progress toward troubleshooting and repairing a complex system.

With thoughtful partitioning, a board may be determined to be good without running all of its individual functional block tests; some functional blocks can be assumed to be good if tests for other functional blocks that depend on them are good.

Through partitioning, the large problem of testing and troubleshooting a complex system can be subdivided into smaller, more easily handled problems.

PROGRAM DEVELOPMENT SEQUENCE 3.3.

There are four levels in programming with the 9100A, as shown in Figure 3-3. Each level is a building block for the next level of programming.

The sequence shown below is the most efficient method of developing programs if you plan to develop both functional testing and GFI troubleshooting capability. This is because the functional block tests in Step 2 can often use the GFI stimulus programs developed in Step 1 to test the outputs of a functional block (See Section 3.5.1 for additional explanation). However, in other situations, you may need to use your 9100A/9105A for functional testing as soon as possible, even before troubleshooting programs can be developed. In this case you may want to do Steps 2 and 3 before doing Step 1.

The four steps of programming are:

1. *Stimulus programs* for nodes are created and *responses* from a known-good UUT are learned. (Sections 3 and 4 of this manual)

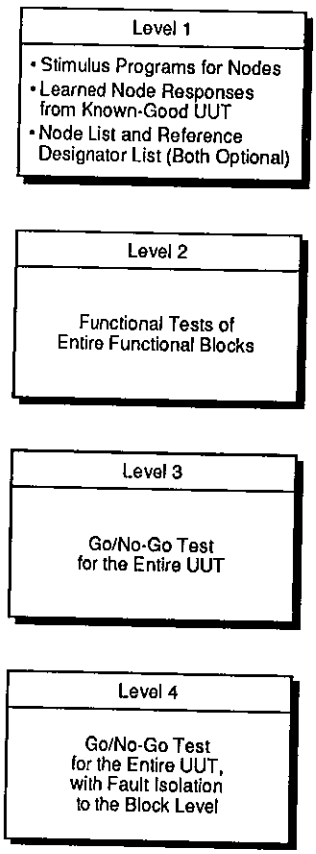


Figure 3-3: Building-Block Programming

If the node list and reference designator list are also created, this level will allow not only testing a node, but also automated backtracing from a bad node to the fault.

2. *Functional tests* of entire functional blocks are created. The *gfi test* command can use your stimulus programs and learned responses for fast, effortless functional tests of these blocks. (Sections 3 and 4 of this manual.)

3. *A UUT go/no-go test* is built from the functional tests of functional blocks. (Section 5 of this manual)

4. *Diagnostic programs* are created by adding fault handlers and *gfi hint* commands to the UUT go/no-go test. The diagnostic program traps faults and initiates tests of functional blocks that may be responsible for the fault, thereby isolating the functional block that is causing the UUT to fail. When the failing output of the block is found, then a GFI hint is generated and GFI will begin backtracing the failing circuitry. (Section 6 of this manual)

After the fourth programming level, the go/no-go test will isolate the failing functional block and then will start GFI troubleshooting (Section 7 of this manual) to backtrace to the bad node or component.

STIMULUS PROGRAMS AND LEARNED RESPONSES 3.4.

Stimulus programs and learned responses constitute the first of the four levels in programmed testing and troubleshooting, as shown in Figure 3-4.

Stimulus programs create predictable node activity so that one or more nodes can be characterized. When properly designed, these programs are usually short and simple. With the 9100A,

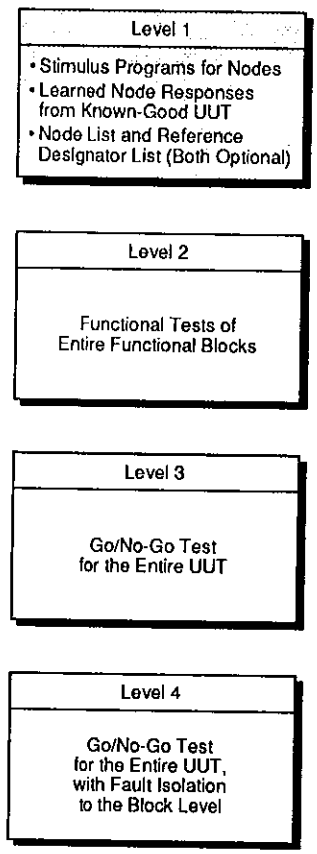


Figure 3-4: Functional Tests for Nodes (Level 1)

the most difficult task related to writing stimulus programs is understanding how the UUT operates.

Learned responses are the responses of a known-good UUT to the stimulus programs. The 9100A/9105A can store these responses from a known-good UUT for use in testing other identical UUTs.

Rules for Stimulus Programs 3.4.1.

Stimulus programs must follow these rules, to ensure that GFI troubleshooting reaches correct conclusions:

- *Measure Outputs.* Use stimulus programs to characterize signal sources (outputs) only.
- *Provide Initialization.* If a circuit ever requires initialization, place an initialization procedure in the stimulus program. The initialization must be performed before the measurement is started. The best place for initialization is near the beginning of the stimulus program.
- *A Separate Program for Each Signal Source on a Node.* Create a separate stimulus program for every signal source (output) on a node. A bidirectional line between two components should have at least two stimulus programs, one for each direction of data flow. Buses should have at least one stimulus program for every component that can output on the bus.
- *A Separate Program for Each Mode of Output.* Create a separate stimulus program for each way that an output is operated in normal UUT operation. For example, if a buffer on an address bus is stimulated by the microprocessor and also by a DMA controller, create two stimulus programs for the outputs of the buffer: one from the microprocessor, and one from the DMA controller.
- *Keep It Simple.* If a stimulus program becomes complex, find a way to split it up into more than one program. For example, consider a PIA chip connected to a data bus and a keypad that can be read through the PIA. The stimulus

program that enables the PIA data lines onto the data bus should initialize registers at the beginning of the program, then the program should read the registers in the PIA chip.

The Flow of Stimulus Across the UUT 3.4.2.

Stimulus programs are unrelated to functional blocks. Functional blocks are only defined to help with functional testing.

Stimuli generally flow from the microprocessor kernel toward the outputs of the UUT. Some stimulus programs may characterize the outputs of many components while other stimulus programs may characterize only a few outputs.

The key to efficient stimulus programs is to begin at some outputs of the microprocessor kernel that can be stimulated. Stimulate these outputs and trace through the circuit to see how many other output nodes can be characterized. Find nodes that have not been characterized, and decide what is needed to stimulate them. Then, see how many nodes are covered. Continue this process until each node is covered by at least one stimulus program.

A good way to keep track of which nodes have been covered is to use a set of colored markers. Using a separate color for each stimulus program, color in a small region around the output nodes which will be stimulated by that program (remember, stimulus programs only apply to signal sources). Even for a complex UUT, the strategy for creating stimulus programs for an entire UUT can be "mapped out" in a few hours. The time spent will promote better software organization and speed up both the writing of stimulus programs and the process of learning the responses.

Keep in mind the rules described in the Section 3.4.1, and remember that some outputs will be characterized by more than one stimulus program.

Stimulus Program Planning 3.4.3.

Stimulus programs and their matching response files are used by the 9100A/9105A Guided Fault Isolation (GFI) to backtrace through a failing circuit in a UUT to find the fault. The stimulus programs exercise a portion of the UUT circuitry in order to produce repeatable activity at circuit nodes to be measured. This activity at each node is measured on a known-good UUT and a characterization of this known-good response is stored in a response file. Each response file stores characterizations of how some circuit nodes on a good UUT perform as a result of its matching stimulus program. There is one response file for every stimulus program.

Each of the fourteen functional blocks in Section 4 includes a figure titled "Stimulus Program Planning." Figure 3-5 shows an example of such a figure.

The purpose of the stimulus program planning diagrams is to illustrate how to design the stimulus programs for a UUT. In general, you should begin the process of creating stimulus programs by identifying outputs from the microprocessor that can be exercised (such as the address bus, data bus, and control lines). Characterize all those nodes that are stimulated, then find some nodes that are not characterized and design stimulus programs to stimulate them. In general, start at the microprocessor and work outwards to the I/O devices. Continue until all nodes in the UUT are characterized.

The left-hand page of Figure 3-5 shows six blocks that represent six stimulus programs and their matching response files. Each of these stimulus program/response file pairs are used to stimulate and characterize nodes in this functional block.

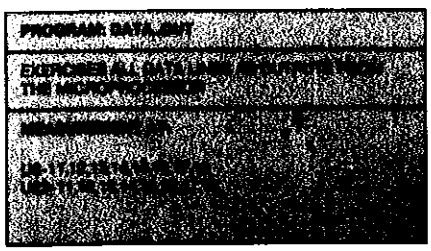
The block for the *addr_out* stimulus program shows that it stimulates the outputs of the address buffers: U16, U2, and U22. As you examine each of the stimulus program planning figures in Section 4 of this manual, you will notice that the *addr_out* stimulus program stimulates nodes in many of these functional blocks. This is because the stimulus programs are not

(This page is intentionally blank.)

Example

Stimulus Program Planning

PROGRAM: ADDR_OUT
EXERCISES ALL ADDRESS LINES FROM THE MICROPROCESSOR
MEASUREMENT AT: U16-19,16,15,12,9,6,5,2 U2-19,16,15,12,9,6,5,2 U22-19,16,15,12,9



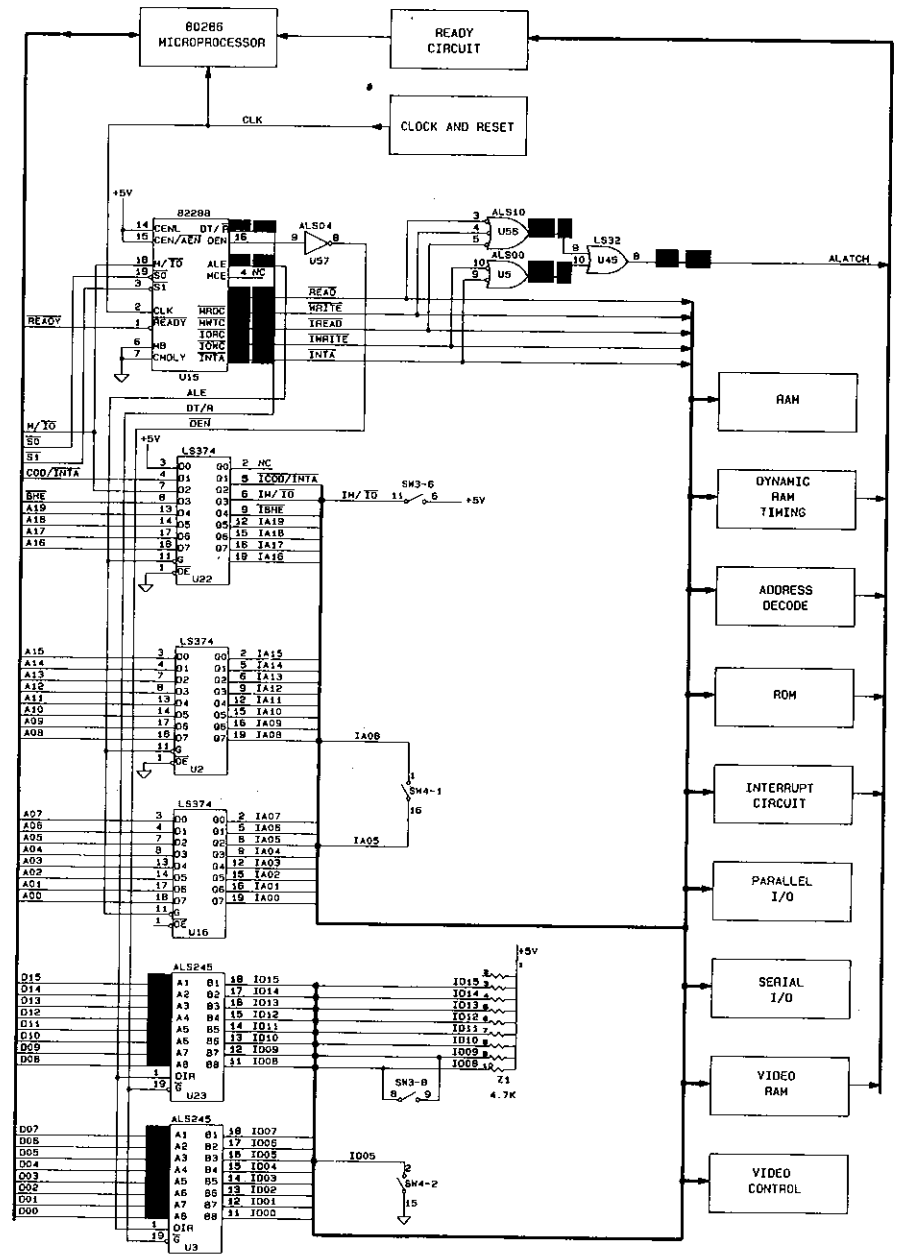


Figure 3-5: Example of Stimulus Program Planning Figure

limited by functional block boundaries and typically will stimulate nodes over several functional blocks.

Figure 3-5 shows that the *data_out* stimulus program stimulates the bidirectional data bus when the microprocessor is sending out data (a write operation). The figure also shows that the *rom1_data* stimulus program is used to stimulate the data bus buffers U3 and U23 when data is flowing into the microprocessor (a read operation).

The other three stimulus programs shown (*ctrl_out1, ctrl_out2,* and *ctrl_out3)* stimulate the control line outputs from the microprocessor and bus controller IC (an 82288 chip); *ctrl_out1* stimulates the control lines using pod data synchronization; *ctrl_out2* stimulates the control lines using pod address synchronization; *ctrl_out3* generates an interrupt acknowledge cycle and stimulates the control lines using interrupt acknowledge synchronization.

When planning the stimulus programs for your UUT, you can use colored pens to map out which outputs in your UUT will be covered by which stimulus programs. You should start with the address signals, data signals, and control signals. After that, you can plan what is required for stimulus programs for other outputs in your UUT, working from the kernel toward the I/O of the UUT.

Suggestions about Stimulus Programs 3.4.4.

The actual stimulus programs used for the Demo/Trainer UUT are listed in Section 4 of this manual. Some stimulus programs stimulate nodes in several functional blocks and other stimulus programs stimulate only a few nodes. The fact that, in Section 4, stimulus program coverage is organized by functional blocks does not imply that the stimulus programs observe functional-block boundaries. Stimulus programs do not care about functional block boundaries and usually will exercise nodes across functional-block boundaries.

Each of these stimulus programs in Section 4 follows a standard form that can be divided into five parts:

- Initialize the circuit and define the measurement device.
- Set up the stimulus and measurement devices.
- Start the measurement.
- Stimulate the circuit.
- Stop the measurement.
- Restore any conditions changed by the setup, above.

Figure 3-6 shows a simple stimulus program with each of the six parts labeled. Circuits that contain programmable components require initialization. Any circuit that needs initialization should have it provided in the stimulus program. This is necessary since there is no way to determine the order in which stimulus programs will be run when GFI or UFI troubleshooting is performed. Therefore, each stimulus program should perform any initialization the circuit needs.

Defining the Measurement Device

Most stimulus programs use the I/O modules and the probe as measurement devices. When GFI or UFI is using the I/O module as a measurement device, a message is displayed which prompts the operator to clip onto the component and to push the Ready button on the clip module. When the operator does this, the 9100A/9105A identifies the I/O module and the side (A or B) being used.

GFI or UFI can tell a stimulus program which device is being used. It is a good idea to write your stimulus programs so that the measurement device name is obtained from GFI or UFI rather than specifying the device name in the stimulus program. Getting the name from GFI or UFI has the advantage that the operator can connect a clip to either side of any of the four I/O modules. The operator can use several I/O modules, each with a

```
program data_bus

   if (gfi control) = "yes" then              ! DEFINE THE MEASUREMENT
      devname = gfi device                    ! DEVICE
   else
      devname = "/mod1"
   end if

   podsetup 'enable ~ready' "off"             ! SET UP THE MEASUREMENT
   podsetup 'report power' "off"              ! AND STIMULUS DEVICES
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   setspace space (getspace space "memory", size "word")
   reset device devname
   sync device devname, mode "pod"
   sync device "/pod", mode "data"

   arm device devname                         ! START THE MEASUREMENT

      rampdata addr 0, data 0, mask $FF       ! STIMULATE THE CIRCUIT
      rampdata addr 0, data 0, mask $FF00

   readout device devname                     ! STOP THE MEASUREMENT

   podsetup 'enable ~ready' "on"              ! RESTORE READY

end program
```

Figure 3-6: Parts of a Stimulus Program

different size of clip, and the stimulus program will still work with any of these configurations.

The stimulus program shown in Figure 3-6 uses the TL/1 *gfi control* command to determine that GFI or UFI is executing the stimulus program. If GFI or UFI is executing the program, the *gfi device* command is used to return the name of the measurement device.

Using the I/O Module as a Stimulus Device

Each I/O module can be used to overdrive a limited number of components. The same I/O module or a different I/O module may be used to measure circuit response.

For example, suppose an I/O module is used to perform a truth-table test of a 7400 NAND gate. The I/O module is clipped to the 7400. Pins 1 and 2 of the 7400 are inputs and pin 3 is the output. The same I/O module drives the inputs and measures CRC signature responses on the output. Each time the pattern is driven on the inputs, the output's CRC signature is sampled.

In this example, the same I/O module is used as the stimulus device and as the measurement device. In some cases, more than one clip is used in stimulating and measuring circuit response. The *gfi device* command returns the device name of the measurement device being used.

The stimulus program should use the *clip* command or the *assoc* command to identify the stimulus device. This command will prompt the operator to clip to the component and push the Ready button on the clip module. Using this method to identify the stimulus device creates a program that allows the operator to use any I/O module for the measurement device and any other I/O module for the stimulus device.

Two steps are necessary to drive a pattern on a set of inputs. First, a *storepatt* command is written for each input pin to be driven. If five inputs are to be driven, five *storepatt* commands are needed. After the patterns are defined by *storepatt*, a

writepatt command is used to clock out all the defined patterns in parallel.

The I/O module has 40 lines. Clips have 14 to 40 pins. Each clip maps to the I/O module lines in a different way. The 40-pin clip is one for one (clip-pin 1 is mapped to I/O module-line 1, etc.). The other clips have a different mappings (shown in appendix B of the *Technical User's Manual).*

The TL/1 commands that involve an I/O module refer to pin numbers in three different ways. These TL/1 commands have a parameter that specifies the device name. If the device name is an I/O module name (such as "/mod1"), any pin numbers in that command will be treated as I/O module line numbers. If the device name is a clip module name (such as "/mod1A"), any pin numbers in that command will be treated as clip module pin numbers. And, if the device name is a reference designator (such as "U14"), any pin numbers in that command will be treated as component pin numbers.

If the device name is a reference designator, the component must have been clipped in response to a request from GFI, or in response to a TL/1 *clip* command prior to being used in an I/O module command.

Consider the example of a 7400 that is to have pins 1 and 2 driven by the I/O module. The reference designator for the 7400 is U3. The following TL/1 commands will perform a truth table test on one gate in the 7400:

```
dev = clip ref "U3", pins 14
storepatt device "U3", pin 1, patt "1010"
storepatt device "U3", pin 2, patt "1100"
arm device dev
    writepatt device dev
readout device dev
```

The *clip* command must be used here to define the I/O module and the I/O module side (A or B) that is clipped to U3. The two *storepatt* commands define the pattern to drive on pins 1 and 2 of U3. Because a reference designator was used as the device name (rather than a clip module name like "/mod1A") in the

storepatt commands, any size of clip can be connected to U3. Suppose a 16-pin clip is connected to U3. The 9100A/9105A knows, from the *clip* command, that the part has 14 pins. As long as pin 1 of the 16-pin clip is placed on pin 1 of the component, the 9100A/9105A will map the pins correctly. GFI and UFI are also able to use clips larger than the component they clip over to measure the response of that component.

The *arm* and *readout* commands start and stop the measurement. Inside the measurement, the *writepatt* command sends the defined patterns to the specified pins. Because the *writepatt* command is surrounded by the *arm* and *readout* commands, a CRC signature can be gathered on the input pins or the output pins of the component, as determined by GFI.

In general, stimulus programs can be written so that any I/O module can be used either for stimulus or measurement. To do this, use the device name returned by the TL/1 *gfi device* command for measurement devices. If the stimulus program uses the I/O module as a stimulus device, use the *clip* statement and reference names for device names in the TL/1 commands (pin-number parameters) that interact with the I/O module.

FUNCTIONAL TESTS 3.5.

Functional tests of blocks are the second of the four modular levels in programming the 9100A, as shown in Figure 3-7. In this second level, tests of functional blocks are created from stimulus programs and response files.

The goal of a functional test is to evaluate the performance of the functional block and to decide whether the entire block is good (passes) or bad (fails). As shown in Figure 3-8, such a test can be divided into the following steps:

1. Initialize the circuits in the block (if necessary).
2. Stimulate the inputs to the block.
3. Measure the outputs from the block.

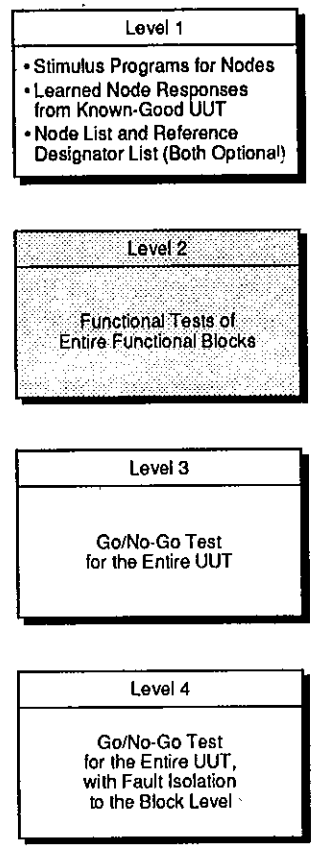


Figure 3-7: Functional Tests for Functional Blocks (Level 2)

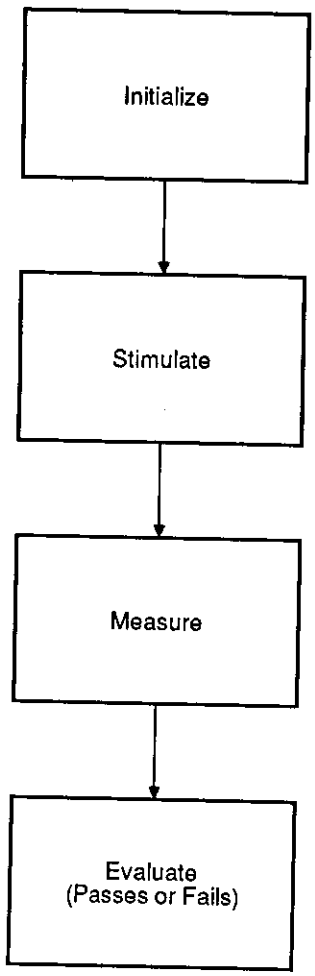


Figure 3-8: Functional Test Elements

4. Evaluate each output and decide whether the output passes or fails. If all outputs pass, the block is good, otherwise it is bad.

Programmed Functional Tests 3.5.1.

Programmed functional tests perform all four functional test steps automatically. There are three basic methods of writing functional tests for each functional block in the UUT:

- *Using the TL/1 built-in functional test commands* - Use for testing the microprocessor bus, RAM, and ROM.
- *Building on stimulus programs* - Use the *gfi test* command to build on stimulus programs and learned responses.
- *Writing TL/1 programs which are independent of GFI* - These programs must perform all four functional test steps.

Using Built-In Functional Test Commands

For some functional blocks, such as the microprocessor bus, ROM, and RAM, you should not use the *gfi test* command. Instead, these blocks can be tested with the built-in TL/1 functional test commands *testbus, testramfast, testramful, and testromfull.*

Building On Stimulus Programs

Stimulus programs and learned responses are used to decide if a node passes or fails. The TL/1 programming language has a command called *gfi test*, which performs Steps 1 through 3 of functional testing and part of Step 4 (see Figure 3-8).

The *gfi test* command tests an entire component (if the I/O module is the measurement device) and returns a passes or fails result. The command runs all stimulus programs associated with all pins on the component and compares the responses to

the learned responses. It returns a "passes" result if all pins on the component are good.

Suppose the buffers of a 24-bit microprocessor address bus are tested as a functional block. If the functional test is written without the *gfi test* command, the test would perform the following operations:

1. Stimulate the address bus.
2. Capture signatures on the 24 address lines.
3. Compare captured signatures with known-good signatures (24 if/then statements).

The same functional test using the *gfi test* command would require only three *gfi test* commands. Using this command decreases the time required to write functional test programs.

Using the *gfi test* command provides an additional important advantage. When it is used, the known-good responses are automatically retrieved from the the 9100A/9105A's response files. Whenever a board is revised, the response files must be updated. If a functional test contains known-good response information built into the program, rather than stored in response files, both the response file and the functional test program must be updated if the board is revised.

You may need to develop a test quickly for just one functional block and avoid writing stimulus programs or learning responses for the entire UUT. In this case, the following procedure will help ensure that the functional test you write will later integrate well into the functional test for the entire UUT:

1. Make a plan for the stimulus programs you will need to cover the entire UUT. This usually takes several hours.
2. Write the stimulus programs needed to test the block in question.

3. Write the functional test for the block using the *gfi test* command wherever possible.

4. After the test for the block is finished, you can continue with the process of writing stimulus programs and learning responses for the rest of the blocks in the UUT.

Functional Tests That Are Independent of GFI

You can also write functional tests that do not require the use of stimulus programs and response files. If so, these tests should also contain the functional test elements shown in Figure 3-8.

Programmed Functional Test Examples 3.5.2.

The programmed functional tests for each functional block in the Demo/Trainer UUT are listed in Section 4 of this manual. The simplicity of these functional tests results from using the *gfi test* command and the built-in test functions.

It is tempting to write a functional test without first writing stimulus programs. However, a penalty is paid for this approach in two ways: it can actually take longer if stimulus programs are not created first, since the 9100A/9105A already has built-in functions to do much of the functional testing once stimulus programs are created. Second, stimulus programs will have to be written anyway before GFI troubleshooting can be used.

The sequence of steps shown in Figure 3-7 will in most cases give you the best results in the shortest time. Each increment of programming investment will result in better performance and productivity.

Keystroke Functional Tests 3.5.3.

A UUT may be tested using only 9100/9105A front panel keystrokes. Keystroke testing also involves each of the four functional test steps (initialization, stimulation, measurement, and evaluation) shown previously in Figure 3-8, but the operator performs these steps rather than having the 9100A/9105A do them with TL/1 programs. If you wish, these steps can be stored in keystroke sequences by using the SEQ key on the operator's keypad.

Each of the fourteen functional blocks in Section 4 has a "Keystroke Functional Test" figure like the example shown in Figure 3-9. The purposes of these figures are the following:

- Show the schematic diagram for that functional block.
- Show the inputs to the functional block from other functional blocks.
- Show the outputs of the functional block to other functional blocks.
- Identify the 9100A/9105A measurement and stimulus devices used to test the block and to identify where those devices are connected.
- Show the expected node response information from performing the functional test sequence for that block.

Figure 3-9 is a typical example of a Keystroke Functional Test figure that you will see for each of the functional blocks described in Section 4 of this manual. In most cases, the functional blocks to the left of the schematic are those which provide input to the functional block shown in schematic form. In most cases, any functional blocks to the right of the schematic are receiving the outputs of the functional block shown in schematic form. The arrows show the direction of the signals between the functional block boxes and the functional block shown in schematic form.

Notice the left-hand page of Figure 3-9. At the top of the page is a box labeled CONNECTION TABLE. The left column of this

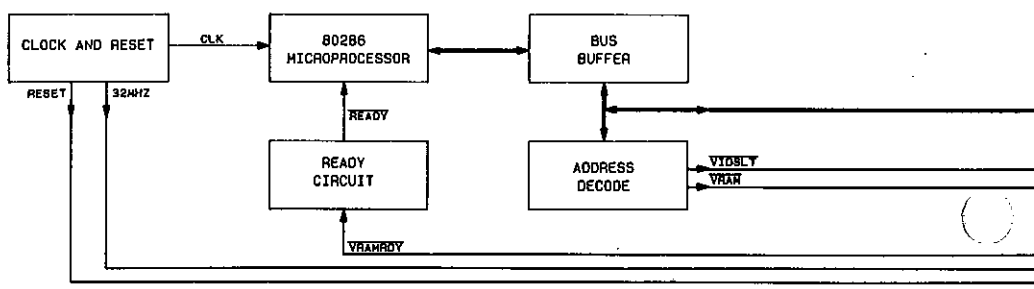
Keystroke Functional Test

CONNECTION TABLE

	MEASUREMENT CONTROL	MEASUREMENT
(NONE)	I/O MOD	I/O MOD
	CLOCK U78-33 START U88-13 STOP U88-13 ENABLE U78-12	U72

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
DAD00	U72-34	34	4155
DAD01	-33	33	3F33
DAD02	-32	32	A65A
DAD03	-31	31	9024
DAD04	-30	30	DE6D
DAD05	-29	29	D6FA
DAD06	-28	28	7AC3
DAD07	-27	27	0477
DAD08	-26	26	E941
DAD09	-25	25	88B8
DAD10	-24	24	60B0
DAD11	-23	23	D869



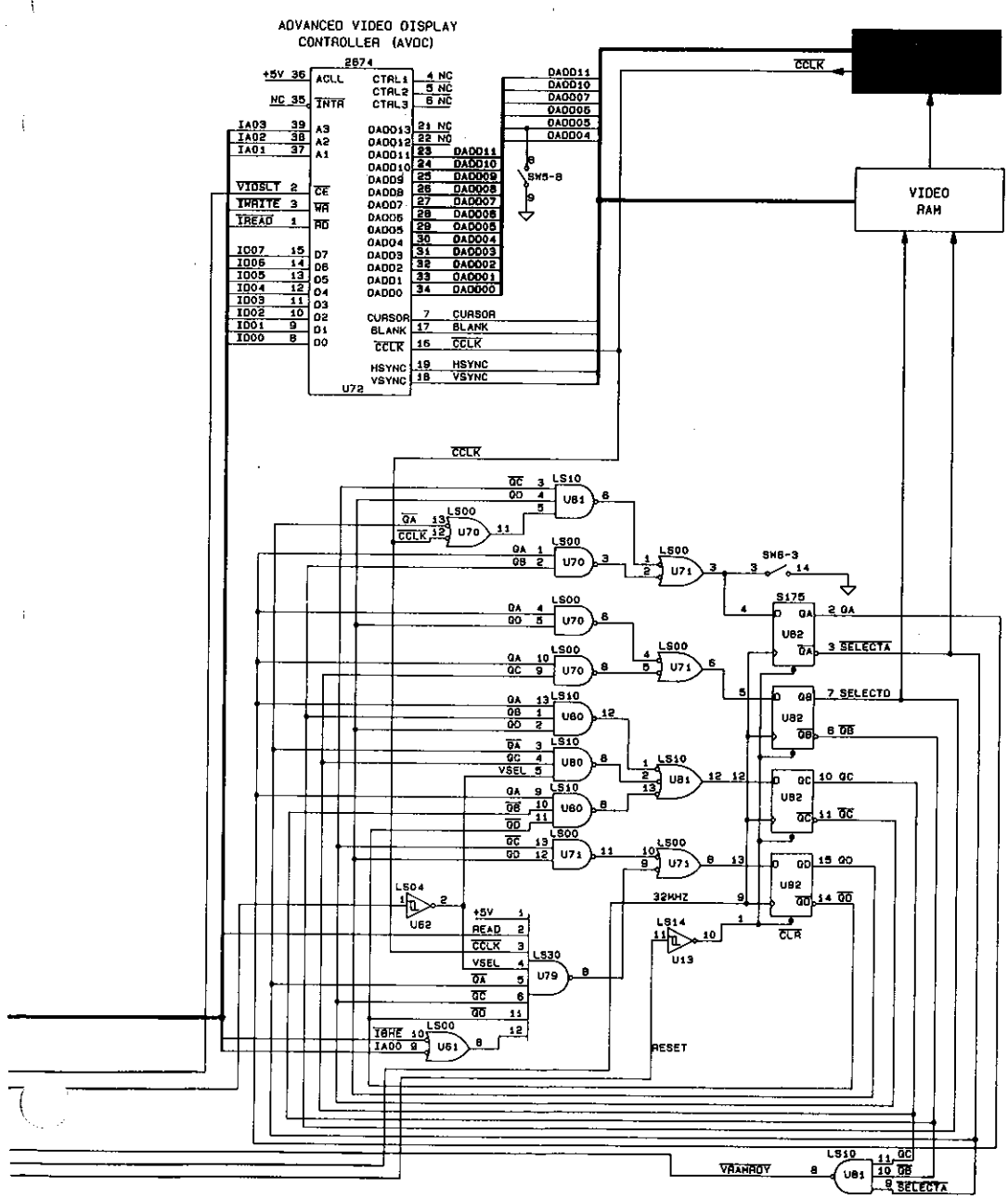


Figure 3-9: Example of Keystroke Functional Test Figure

table, labeled STIMULUS, shows what 9100A/9105A device is used to provide stimulus to the functional block shown in schematic form and where the connection is made. In the example shown in Figure 3-9, no stimulus is provided because this diagram is part of the video circuit and, once initialized, the video circuit constantly runs with no additional stimulus. In many of the keystroke functional test diagrams in Section 4, the STIMULUS column will indicate that the pod or I/O module is used.

The right column of the CONNECTION TABLE, labeled MEASUREMENT, shows which 9100A/9105A device is used to measure circuit response for the Keystroke Functional Test. The measurement device can be the probe, the pod, or an I/O module. This column also shows the components or nodes in the circuit that are to be measured.

When the I/O module is the measurement device and its external control lines are used, a third column, labeled MEASUREMENT CONTROL, shows where to connect the START, STOP, CLOCK, and ENABLE lines.

The RESPONSE TABLE shows the names of the signals to be measured, the component and pin numbers to be measured, the corresponding pin numbers used by the I/O module, and the known-good measurement value for each signal.

Section 4
Functional Block Test and Troubleshooting Examples

This section is organized into fifteen sub-sections. The first fourteen sub-sections each contain the following information:

- General discussion of a kind of functional block.
- Testing and troubleshooting approaches.
- Keystroke testing procedure for Demo/Trainer UUT.
- Functional test program listing for Demo/Trainer UUT.
- Stimulus programs and responses for troubleshooting.
- Summary of solution showing all files and programs needed to test and troubleshoot the functional block.

The last sub-section covers types of circuits not found in the Demo/Trainer UUT and is therefore organized differently than the above.

For the purpose of learning how the 9100A/9105A works, each of the fourteen functional blocks can be considered to be a self-contained portion of the UUT. The Summary of Solution page at the end of each sub-section shows all of the files required to test or troubleshoot that functional block.

Only a subset of all the functional blocks in a UUT needs testing to determine whether the UUT is good or bad. This is because testing the major functional blocks indirectly tests the other blocks as well. (See Section 5 for more details on functional

testing strategy for a complete UUT). For the Demo/Trainer UUT, testing the following major functional blocks will be sufficient for a UUT go/no-go test functional test:

- Microprocessor Bus.
- ROM.
- RAM.
- Parallel I/O.
- Serial I/O.
- Video Output.

The remaining functional blocks covered in this section are useful for troubleshooting the UUT if it fails the go/no-go UUT functional test:

- Dynamic RAM Timing.
- Video Control.
- Video RAM.
- Bus Buffer.
- Address Decode.
- Clock and Reset.
- Interrupt Circuit.
- Ready Circuit.

You will find that the Dynamic RAM Timing, Video Control, and Video RAM functional blocks come from subdividing the RAM and Video blocks into smaller-size blocks.

MICROPROCESSOR BUS FUNCTIONAL BLOCK 4.1.

Test Access to the Microprocessor Bus 4.1.1.

The term "test access" refers to the point at which the pod connects to the Unit Under Test (UUT). In most cases, a UUT's microprocessor or microcontroller is replaced in its socket by the pod, but this is not always the case. For example, if the microprocessor is soldered in, the UUT can be designed to allow a bus-cycle emulation pod to access the bus through a test connector.

The test access allows the 9100/9105A pod to perform reads and writes on the microprocessor bus. The pod can selectively ignore inputs which normally would go directly to the microprocessor. Thus, any faults that would stop the microprocessor can be ignored by the pod, and testing can proceed as though the microprocessor were in a good circuit and functioning properly.

The pod uses microprocessor bus emulation as the primary means of testing and troubleshooting. It can generate stimuli to the UUT and capture the responses in conjunction with other 9100/9105A stimulus and measurement devices, thereby providing excellent troubleshooting capability for all microprocessor signals. The pod can perform basic microprocessor read and write operations, various stimulus functions built from multiple reads and writes, and built-in tests such as bus, RAM, and ROM. The pod also verifies that the microprocessor power supply is within tolerance, and that all power supply pins are connected.

A little foresight in the design of test access can make testing much easier. Here are some general guidelines to facilitate testing:

- Provide clearance around all devices. This allows access for the pod connectors (to replace the microprocessor or plug in a test socket), for a component extraction tool (if components are hard to remove, especially pin-grid array

(PGA) types), and for I/O module clips (especially if adjacent components must be clipped simultaneously).

- Provide some means to access the microprocessor bus if the microprocessor is soldered in. An additional microprocessor socket or card edge connector can provide this access. Consider providing some form of test access even though the factory or service center may use test fixturing, since this allows testing in field situations where no test fixturing is available.
- Use resistors to the power supply or ground to establish static logic levels on unused inputs instead of directly connecting inputs to power supplies or ground. This allows the 9100/9105A to drive these inputs.
- If there are microprocessor inputs that will force most of the microprocessor outputs to a high-impedance state, design the UUT so that the 9100/9105A can drive these inputs.
- If there are microprocessor outputs that cannot be placed in a high-impedence state, design the UUT so that these outputs are buffered and the buffer outputs can be turned off or overdriven by the 9100/9105A.
- Allow the UUT clock to be suppressed to permit the UUT to operate with an external clock.
- Ensure that vendors' specifications for load and timing margins are not violated and, if possible, allow for a further margin.
- Design so that all signals at a ROM chip can be latched by the I/O module with DATA synchronization.
- Pull up all lines carrying data signals to a logic 1 through resistors.

Considerations for Testing and Troubleshooting

Kernel Testing

The combination of the microprocessor, ROM and RAM is collectively referred to as the kernel. The primary advantage of Fluke test and troubleshooting equipment for microprocessor-based UUTs over equipment from other vendors is its ability to troubleshoot dead kernels.

If any part of the kernel malfunctions, very little else works properly. One basic strategy is to test the kernel first, then test the other functional blocks surrounding the kernel.

The 9100/9105A has a comprehensive built-in test for the unbuffered microprocessor bus. This bus test is a series of reads and writes at different addresses while monitoring microprocessor outputs for faults. The bus test is described in detail in Section 6.2.1 of the *Technical User's Manual*. With this bus test, the 9100/9105A can determine stuck or tied lines on all outputs from the microprocessor bus. During a bus test, active interrupts or forcing signals that cause the microprocessor to malfunction will be intercepted by the pod and reported to the 9100/9105A unless they have been specifically disabled with the *podsetup* command in TL/1 or the SETUP POD command on the operator's keypad. The bus test will also report a bad power supply or an inactive clock.

Figure 4-1 summarizes the major conditions reported by the bus test. Faults such as stuck bus lines, missing clocks, and low UUT voltages must be cleared before further testing can proceed.

Finding the source of bus faults may be complicated by multiple bus-master and intervening buffers. For example, a buffer may be loading the bus because its enable line is asserted due to faulty circuitry back several logic gates from the bus. If there are several bus masters, it may be unclear where the fault is. Bus masters may be identified as **masters* in the node list. The

Signal Group	*Condition*	*Example Message*
Address Lines	stuck, tied	addr line A9 pod pin 22 stuck high
Data Lines	stuck, tied	data line D8 pod pin 37 tied data line D9 pod pin 39 tied
Control Lines	stuck, tied	control line HLDA pod pin 65 not drivable
Interrupt Lines	active	interrupt ~INTR pod pin 57 active
Forcing Lines	active	forcing signal RESET pod pin 29 active
Clock	inactive	pod timeout bad UUT clock
Power Lines	out of tolerance	bad UUT power supply

Figure 4-1: Conditions Reported by the BUS TEST

**masters* identification, combined with independent stimulus programs for each bus master, assist GFI in backtracing faults identified on buses.

For more information on **masters*, stimulus programs, and response files, see Section 7 of this manual and Section 5.5 of the *Programmer's Manual.*

Basic Bus Cycles

It is often useful to perform a series of reads and writes to verify proper operation of basic bus cycles. To do this, you need the address map of the UUT. You can verify bus-cycle operation with reads from and writes to RAM, ROM, or other memory-mapped VLSI devices such as PIAs, DMA controllers, SCSI controllers, and UARTs. If your UUT's microprocessor allows transfers of different data widths (byte, word, long word), transfers with these different data widths should be verified.

If reads do not return the correct data when no major bus faults are indicated by BUS test, try the built-in RAM test or ROM test. RAM test checks the ability to read and write all RAM cells specified in the address range. ROM test checks the ability to read from ROM and verifies the ROM signature. Other kernel-related functional blocks, such as Address Decode, Bus Buffers, or Ready circuitry should also be tested, as described later in Section 4 of this manual.

Synchronization Modes

When you are troubleshooting faults related to bus cycles, it is useful to synchronize the probe or I/O module to pod operations. The pod itself can be synchronized to different parts of the bus cycle that may be appropriate for a particular test. For example, a microprocessor with multiplexed address and data will output the address only during the first part of a bus cycle. To test for

address faults, an I/O module or the probe can be synchronized to the address using these TL/1 *sync* commands:

```
sync device "/pod", mode "addr"
sync device "/mod1", mode "pod"
```

or from the operator's keyboard using the SYNC key:

```
SYNC I/O MOD 1 TO POD ADDR
```

With the above synchronization, the I/O module can capture address or other information in functional blocks related to the address. In a similar way, the probe or I/O module can be synchronized to data, or to other microprocessor-specific bus-cycle phases implemented by the pod.

Other Microprocessor Cycles

Other microprocessor cycles may be exercised as part of the microprocessor functional block, such as interrupts, bus exchanges, DMA transfers, or coprocessor cycles. Usually, however, implementation of these cycles involves circuitry that is complex enough to be partitioned separately. Here are a few considerations to keep in mind when testing:

- **Interrupts** are reported by the pod as "active interrupt lines". When a RUN UUT command is entered at the operator's keypad, control is returned to the microprocessor. The operator should be sure that the software needed to set interrupt priorities and handle interrupts is present so that RUN UUT operates properly. Some designs employ a watchdog timer, which asserts a non-maskable interrupt or reset unless the microprocessor performs a write within a certain period of time. When you use pod breakpoints, the watchdog timer should be disabled, the pod should be set up to ignore the watchdog timer output, or software should be written to handle the interrupt.

- **Bus Exchanges** take place when the microprocessor gives control of its bus to a requesting component. Pods allow this capability to be enabled or disabled. When enabled, the pod will grant bus requests just as the microprocessor would. The pod may appear to take an abnormally long time to perform certain tests, such as RAM test, if other components take control of the bus or if a fault condition causes bus requests. Disabling bus requests will command the pod not to grant the bus request and will cause bus requests to show up as forcing signal conditions. If the RUN UUT command is entered at the keypad, control is returned to the microprocessor and the bus request line will be re-enabled. Further troubleshooting with RUN UUT may require that the line be physically disabled.
- **DMA controllers** are integrated with some microprocessors, such as the 64180. The DMA channels operate semi-autonomously and interact with the bus exchange capability. Cautions similar to those used with bus exchanges should be used for DMA channels.
- **Dynamic RAM Refresh** capability is included on some microprocessors, such as the Z80 and the 64180. At regular intervals, a refresh cycle is performed and an address is placed on the address bus. The Z80 and 64180 have refresh signals (RFSH and REF, respectively) which indicate when refresh activity occurs. The frequency of these signals may be monitored with the probe to determine if refresh is working properly.
- **Coprocessors** work in conjunction with some 16- and 32-bit microprocessors. These coprocessors usually have a unique set of signals which control the transfer of data with the main processor. Inputs are called status lines and may be read and reported by the pod. Outputs are called control lines and may be written to check drivability or to send information to the coprocessor.

Other Input and Output

There are other types of inputs and outputs specific to each microprocessor which do not fall into the four basic classifications, address, data, control, and status. These are classified as miscellaneous and include signal types such as bit/parallel, serial, and analog I/O. Each pod treats these lines in a manner appropriate to the specific microprocessor. Refer to the particular pod manual for information on how to handle these signal types.

Microprocessor Bus Example 4.1.3.

The Demo/Trainer UUT uses an 80286 microprocessor, which has a 16-bit data bus and a 24-bit address bus. The Demo/Trainer UUT uses only the least significant 20 bits of the address bus.

The 80286 microprocessor remains in the UUT at all times, so the Demo/Trainer UUT includes a test access socket to provide access to the microprocessor bus. Most of the lines in the test access socket are directly connected to the microprocessor, although a few lines such as HOLD and HOLDA are buffered with three-state buffers. The test access switch, S5, selects either the 80286 microprocessor in the pod or the 80286 microprocessor on the UUT to control operation of the UUT.

Keystroke Functional Test 4.1.4.

Use the BUS TEST key to enter the following command:

```
TEST BUS AT ADDR 0
```

The above command is the entire procedure; the Microprocessor Bus functional block (Figure 4-2) can be tested fully with this single test.

The microprocessor bus test is built-in. It is convenient to run first because:

- It's easy.
- It's fast.
- It provides excellent diagnostic information.
- A bus fault will cause almost all other functional tests to fail, so it should be tested first.

The bus test uncovers all drive problems that may occur at the microprocessor socket. These faults will cause other tests to fail, but the diagnostics for bus faults are best with BUS TEST. If a fault is uncovered, a message will be displayed to the operator. See Appendix F in the *Technical User's Manual* for a list of fault messages.

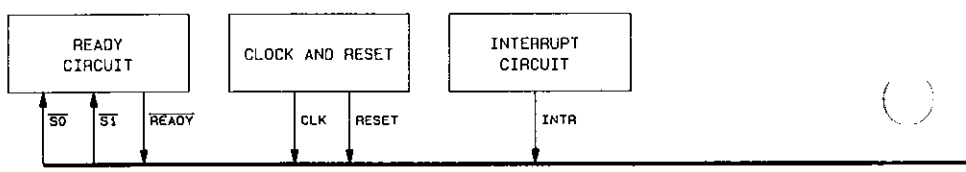
Keystroke Functional Test

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	PQD TEST ACCESS SOCKET

RESPONSE TABLE

(BUILT-IN RESPONSE MEASUREMENT)



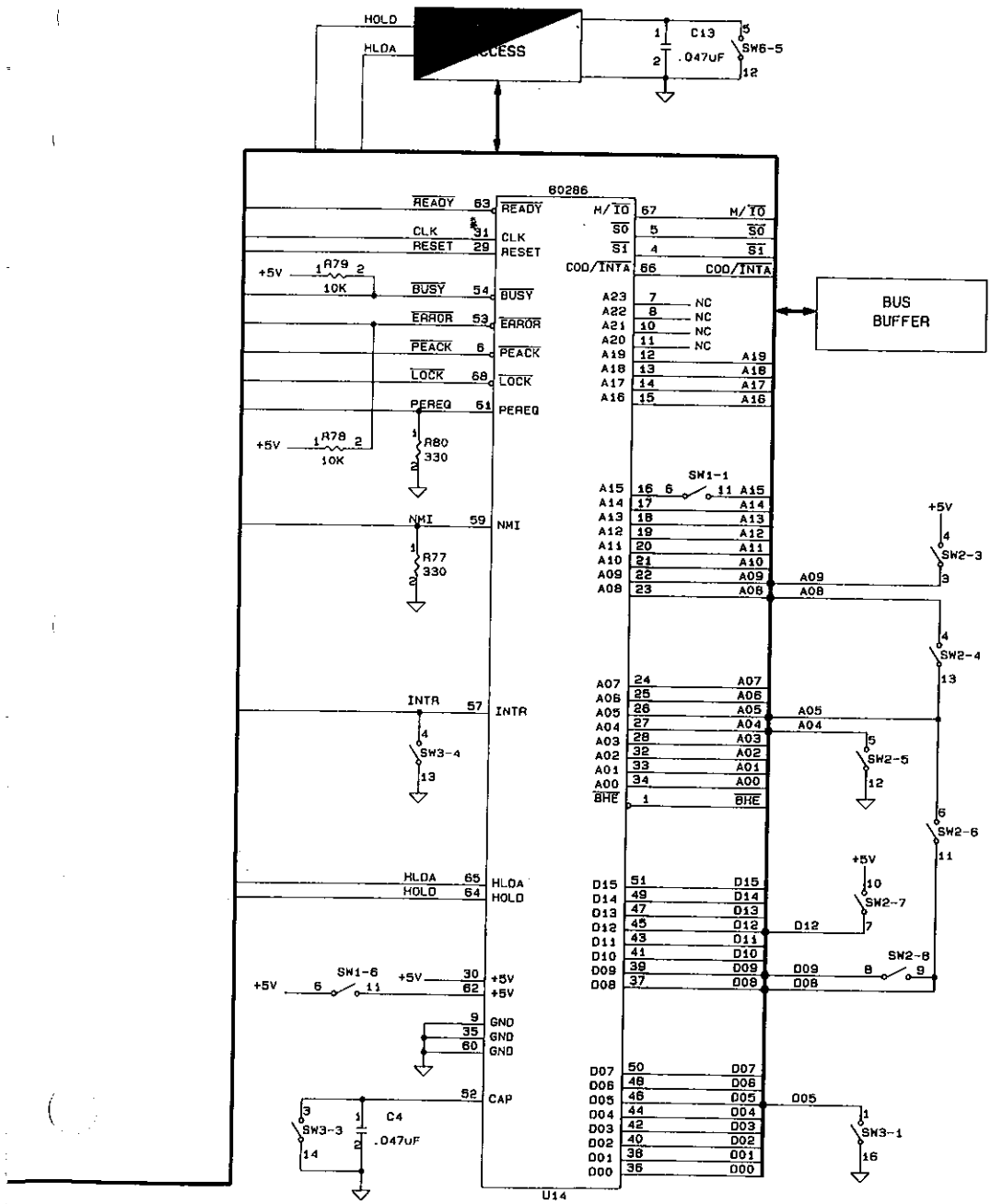


Figure 4-2: Microprocessor Bus Functional Test

Programmed Functional Test 4.1.5.

The Demo/Trainer UUT is determined to be good if functional tests for the Microprocessor Bus, ROM, RAM, Parallel I/O, Serial I/O, and Video Output functional blocks all pass. In order to make the testing as efficient as possible, the buffered bus, address decode, and ready circuitry should be exercised early in the testing. Furthermore, this testing should happen quickly, minimizing the amount of clipping of I/O modules to components.

To meet these goals, the Microprocessor Bus functional test program, *test_bus*, checks the microprocessor bus up to the buffers and also performs an access to every decoded address space (such as ROM, RAM, or Video I/O). These accesses indirectly check the Buffered Bus, Address Decode, Ready, and Interrupt Circuit functional blocks. If a ready or active interrupt problem exists, these accesses to the decoded address spaces will result in an improper ready or active interrupt condition that can be detected by the test.

The *test_bus* program also performs a check for bus contention. Bus contention occurs when a component continually outputs onto the data bus and it is usually caused by faulty enable inputs into a component. The *test_bus* program detects bus contention by reading at a spare address location, which is decoded and can be read from but has no component located at that address to output data onto the data bus. In normal operation, only high bits (logic 1s) are returned on the data bus when the spare address is read. When bus contention drives data bits low, the read at the spare address will detect the problem. In order to detect bus contention that drives data bits high, the *test_bus* program writes all-zero data to RAM and then reads the RAM. If the data read is not all-zero, either the RAM is bad or there is bus contention. To make sure the problem is bus contention, the *test_bus* program reads from two other components on the data bus that are decoded separately. The *test_bus* program uses the ROMs from bank zero and the ROMs from bank 1. If both ROM banks read zero data correctly, the problem is assumed to be a RAM problem (when bus contention occurs, most of the components on the bus will fail). When both ROM banks read

zero data correctly, the *test_bus* program concludes that the problem is not bus contention and leaves further fault isolation to a later test.

If a bus contention problem is detected, a separate bus contention test program called *tst_conten* is executed (see Appendix C for a listing of this program). The *tst_conten* program tests the enable lines for each component that is connected to the data bus. All other information about good or bad data and address lines is ignored by the bus contention program.

The entire *test_bus* functional test runs quickly, but it detects most kernel faults not in the RAM or ROM components.

```
program test_bus

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the Microprocessor Bus.                               !
!                                                                          !
! This program tests the unbuffered microprocessor bus, performs an        !
! access at each decoded address of the buffered bus, and checks the       !
! data bus for bus contention (where a component outputs onto the data     !
! bus at incorrect times).   If bus contention is detected then the        !
! program TST_CONTEN is executed.  TST_CONTEN checks for incorrect         !
! enable line conditions on all the components on the buffered data bus.!
!                                                                          !
! TEST PROGRAMS CALLED:                                                    !
!    tst_conten (addr, data_bits)            Test for bus contention on    !
!                                            the data bus by checking the  !
!                                            enable lines of all devices   !
!                                            on the data bus.              !
!                                                                          !
! Local Constants:                                                         !
!    ZERO_AT_ROM0                            Address of zero data in ROM0  !
!    ZERO_AT_ROM1                            Address of zero data in ROM1  !
!    IO_BYTE                                 I/O BYTE address specifier    !
!    MEM_WORD                                MEMORY WORD address specifier !
!                                                                          !
! Local Variables Modified:                                                !
!    x                                       value returned from a read    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

 declare numeric ZERO_AT_ROM0 = $E002A !Location in ROM0 where 0 exists
 declare numeric ZERO_AT_ROM1 = $F0022 !Location in ROM1 where 0 exists
```

```
! Setup Statements

   podsetup 'enable ~ready' "on"
   podsetup 'report forcing' "on"
   IO_BYTE = getspace space "i/o", size "byte"
   MEM_WORD = getspace space "memory", size "word"

! Test the Unbuffered Microprocessor Bus.

   testbus addr 0

! Test the Extended Microprocessor Bus and Address Decoding.

   setspace (MEM_WORD)
   read addr 0                    ! RAM BANK 0
   read addr $10000               ! RAM BANK 1
   write addr $20000, data 0      ! VIDEO RAM (write only)
   read addr $30000               ! INTERRUPT POLL
   read addr $E0000               ! ROM BANK 0
   read addr $F0000               ! ROM BANK 1
   setspace (IO_BYTE)
   read addr 0                    ! VIDEO SELECT
   read addr $2000                ! RS232 SELECT
   read addr $4000                ! PIA SELECT

! Test for Bus Contention driving lines low by accessing unused address space

   setspace (MEM_WORD)
   x = read addr $50000           ! SPARE-2 ADDRESS SPACE
   if x <> $FFFF then
      execute tst_conten( $50000, cpl(x) and $FFFF)
      return
   end if


! Test for Bus Contention driving lines high by reading and writing RAM
! If failure then check for bad RAM by reading zeros from 2 other devices.

   write addr 0, data 0           ! WRITE and READ RAM addr 0
   x = read addr 0                ! If fails then check for bad RAM
   if x <> 0 then                 ! by reading 0's at ROM0 and ROM1
      if (read addr ZERO_AT_ROM0) <> 0 then
         if (read addr ZERO_AT_ROM1) <> 0 then
            execute tst_conten( 0, x)
            return
         end if
      end if
   end if

end program
```

Stimulus Programs and Responses 4.1.6.

Stimulus programs are TL/1 programs that are executed by GFI for the purpose of troubleshooting faulty circuits. A stimulus program response file should be associated with each stimulus program in order to store the known-good response for each node to be stimulated by the stimulus program. In this functional block, the microprocessor is the only component and its outputs are stimulated in three groups: address lines, data lines, and control lines.

Figure 4-3 is the stimulus program planning diagram for the Microprocessor Bus functional block. It shows three stimulus programs that are used to exercise the outputs in the microprocessor functional block. These stimulus programs (and their associated response files by the same name) exercise and characterize nodes to be measured in the Microprocessor Bus functional block and in other functional blocks as well.

There are several rules for stimulus programs and response files. One is that only outputs are characterized. Another is that data must be characterized while flowing in only one direction. Therefore, the *data_out* stimulus program measures only data coming out from the microprocessor. Other stimulus programs will measure data coming in to the microprocessor.

After the stimulus program planning diagram, the stimulus programs, and the response files, there is a summary page in the form of a UUT Directory. It shows the entire set of stimulus programs, response files, and other files needed to perform testing and troubleshooting on this functional block. The summary page also shows where each of the stimulus programs and response files can be found in this manual. You will notice that each stimulus program and its associated response file (with the same name) are shown in only one location, although the pair will often be used with more than one functional block.

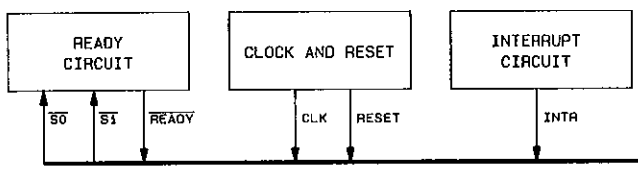
Stimulus Program Planning

PROGRAM: ADDR_OUT
EXERCISES ALL ADDRESS LINES FROM THE MICROPROCESSOR
MEASUREMENT AT: U14-1 U14-34,33,32,28,27,26,25,24 U14-23,22,21,20,19,18,17,16 U14-15,14,13,12

[redacted]

PROGRAM: DATA_OUT
EXERCISES ALL DATA LINES AS OUTPUTS FROM THE MICROPROCESSOR
MEASUREMENT AT: U14-36,38,40,42,44,46,48,50 U14-37,39,41,43,45,47,49,51

[redacted]



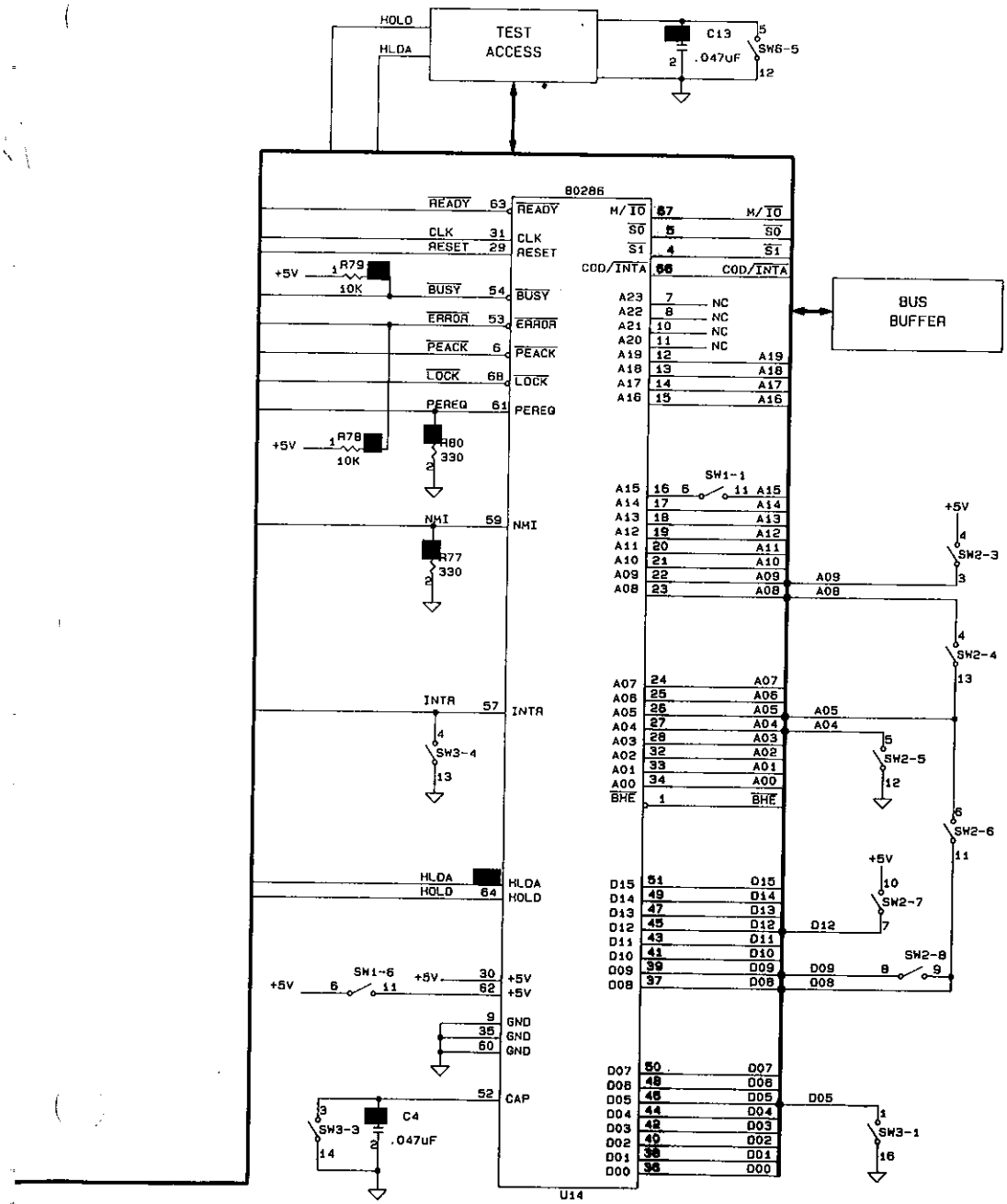


Figure 4-3: Microprocessor Bus Stimulus Program Planning

```
program addr_out

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to wiggle all address lines from the uP.              !
!                                                                        !
! Stimulus programs and response files are used by GFI to back-trace     !
! from a failing node.  The stimulus program must create repeatable UUT  !
! activity and the response file contains the known-good responses for   !
! the outputs in the UUT that are stimulated by the stimulus program.    !
!                                                                        !
! This stimulus program is one of the programs which creates activity    !
! in the kernel area of the UUT.  These programs create activity with    !
! or without the ready circuit working properly.  Because of this, all   !
! the stimulus programs in the kernel area must disable the READY input  !
! to the pod, then perform the stimulus, then re-enable the READY input  !
! to the pod.  The 80286 microprocessor has a separate bus controller;   !
! for this reason, disabling ready and performing stimulus can get the   !
! bus controller out of synchronization with the pod.  Two fault         !
! handlers trap pod timeout conditions that indicate the bus controller  !
! is out of synchronization.  The recover() program is executed to       !
! resynchronize the bus controller and the pod.                          !
!                                                                        !
! TEST PROGRAMS CALLED:                                                  !
!    recover     ()                        The 80286 microprocessor has a!
!                                          bus controller that is totally!
!                                          separate from the pod.  In    !
!                                          some cases the pod can get out!
!                                          of sync with the bus control- !
!                                          ler.  The recover program     !
!                                          resynchronizes the pod and the!
!                                          bus controller.               !
!                                                                        !
! GRAPHICS PROGRAMS CALLED:                                              !
!    (none)                                                              !
!                                                                        !
! Local Variables Modified:                                              !
!    devname                                     Measurement device      !
!                                                                        !
! Global Variables Modified:                                             !
!    recover_times                         Reset to Zero                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-4: Stimulus Program *(addr_out)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   recover_times = 0

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
       devname = gfi device
   else
       devname = "/mod1"
   end if
   print "Stimulus Program ADDR_OUT"

! Set addressing mode and setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   mem_byte = getspace space "memory", size "byte"
   setspace( mem_byte )
   reset device devname
   sync device devname, mode "pod"
   sync device "pod", mode "addr"

! Present stimulus to UUT.

   arm device devname              ! Start response capture.
      rampaddr addr 0, mask $1F
      rampaddr addr 0, mask $1F0
      rampaddr addr 0, mask $1F00
      rampaddr addr 0, mask $1F000
      rampdata addr $20000, data 0, mask $F0
      rampaddr addr $30000, mask $F00
      rampaddr addr $E0000, mask $1F000
   readout device devname          ! End response capture.

   podsetup 'enable ~ready' "on"
end program
```

Figure 4-4: Stimulus Program *(addr_out) - continued*

STIMULUS PROGRAM NAME: ADDR_OUT
DESCRIPTION: SIZE: 1,194 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U16-19	I/O MODULE	DEB8	1	0	TRANS		
U16-16	PROBE	4A68	1	0	TRANS		
U16-16	I/O MODULE	4A68	1	0	TRANS		
U16-15	PROBE	421D	1	0	TRANS		
U16-15	I/O MODULE	421D	1	0	TRANS		
U16-12	PROBE	BFDC	1	0	TRANS		
U16-12	I/O MODULE	BFDC	1	0	TRANS		
U16-9	PROBE	113E	1	0	TRANS		
U16-9	I/O MODULE	113E	1	0	TRANS		
U16-6	I/O MODULE	8F00	1	0	TRANS		
U16-5	I/O MODULE	8300	1	0	TRANS		
U16-2	I/O MODULE	B300	1	0	TRANS		
U2-19	I/O MODULE	AED2	1	0	TRANS		
U2-16	I/O MODULE	88CD	1	0	TRANS		
U2-15	I/O MODULE	8296	1	0	TRANS		
U2-12	I/O MODULE	3B90	1	0	TRANS		
U2-9	I/O MODULE	09E8	1	0	TRANS		
U2-6	I/O MODULE	0D9C	1	0	TRANS		
U2-5	I/O MODULE	56D3	1	0	TRANS		
U2-2	I/O MODULE	9CA7	1	0	TRANS		
U14-1	PROBE	60CD	1	0	TRANS		
U14-1	I/O MODULE	60CD	1	0	TRANS		
U14-34	PROBE	DEB8	1	0	TRANS		
U14-34	I/O MODULE	DEB8	1	0	TRANS		
U14-33	PROBE	4A68	1	0	TRANS		
U14-33	I/O MODULE	4A68	1	0	TRANS		
U14-32	PROBE	421D	1	0	TRANS		
U14-32	I/O MODULE	421D	1	0	TRANS		
U14-28	PROBE	BFDC	1	0	TRANS		
U14-28	I/O MODULE	BFDC	1	0	TRANS		
U14-27	PROBE	113E	1	0	TRANS		
U14-27	I/O MODULE	113E	1	0	TRANS		
U14-26	PROBE	8F00	1	0	TRANS		
U14-26	I/O MODULE	8F00	1	0	TRANS		
U14-25	PROBE	8300	1	0	TRANS		
U14-25	I/O MODULE	8300	1	0	TRANS		
U14-24	PROBE	B300	1	0	TRANS		
U14-24	I/O MODULE	B300	1	0	TRANS		
U14-23	PROBE	AED2	1	0	TRANS		
U14-23	I/O MODULE	AED2	1	0	TRANS		
U14-22	PROBE	88CD	1	0	TRANS		
U14-22	I/O MODULE	88CD	1	0	TRANS		
U14-21	PROBE	8296	1	0	TRANS		
U14-21	I/O MODULE	8296	1	0	TRANS		

(continued on the next page)

Figure 4-5: Response File *(addr_out)*

U14-20	PROBE	3B90	1 0	TRANS
U14-20	I/O MODULE	3B90	1 0	TRANS
U14-19	PROBE	09E8	1 0	TRANS
U14-19	I/O MODULE	09E8	1 0	TRANS
U14-18	PROBE	0D9C	1 0	TRANS
U14-18	I/O MODULE	0D9C	1 0	TRANS
U14-17	PROBE	56D3	1 0	TRANS
U14-17	I/O MODULE	56D3	1 0	TRANS
U14-16	PROBE	9CA7	1 0	TRANS
U14-16	I/O MODULE	9CA7	1 0	TRANS
U14-15	PROBE	8E87	1 0	TRANS
U14-15	I/O MODULE	8E87	1 0	TRANS
U14-14	PROBE	A70C	1 0	TRANS
U14-14	I/O MODULE	A70C	1 0	TRANS
U14-13	PROBE	3951	1 0	TRANS
U14-13	I/O MODULE	3951	1 0	TRANS
U14-12	PROBE	3951	1 0	TRANS
U14-12	I/O MODULE	3951	1 0	TRANS
U22-19	I/O MODULE	8E87	1 0	TRANS
U22-16	I/O MODULE	A70C	1 0	TRANS
U22-15	I/O MODULE	3951	1 0	TRANS
U22-12	I/O MODULE	3951	1 0	TRANS
U22-9	I/O MODULE	60CD	1 0	TRANS
U57-4	I/O MODULE	8724	1 0	TRANS

Figure 4-5: Response File *(addr_out) - continued*

```
program data_out

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM for data bus buffers U3 and U23.                          !
!                                                                            !
! Stimulus programs and response files are used by GFI to backtrace         !
! from a failing node.  The stimulus program must create repeatable UUT     !
! activity and the response file contains the known-good responses for      !
! the outputs in the UUT that are stimulated by the stimulus program.       !
!                                                                            !
! This stimulus program is one of the programs which creates activity       !
! in the kernel area of the UUT.  These programs create activity with       !
! or without the ready circuit working properly.  Because of this, all      !
! the stimulus programs in the kernel area must disable the READY input     !
! to the pod, then perform the stimulus, then re-enable the READY input     !
! to the pod.  The 80286 microprocessor has a separate bus controller;      !
! for this reason, disabling ready and performing stimulus can get the      !
! bus controller out of synchronization with the pod.  Two fault            !
! handlers trap pod timeout conditions that indicate the bus controller     !
! is out of synchronization.  The recover() program is executed to          !
! resynchronize the bus controller and the pod.                             !
!                                                                            !
! TEST PROGRAMS CALLED:                                                      !
!    recover     ()                      The 80286 microprocessor has a!
!                                        bus controller that is totally!
!                                        separate from the pod.  In    !
!                                        some cases the pod can get out!
!                                        of sync with the bus control- !
!                                        ler.  The recover program     !
!                                        resynchronizes the pod and the!
!                                        bus controller.               !
!                                                                            !
! GRAPHICS PROGRAMS CALLED:                                                  !
!    (none)                                                                  !
!                                                                            !
! Local Variables Modified:                                                  !
!    devname                             Measurement device                 !
!                                                                            !
! Global Variables Modified:                                                 !
!    recover_times                       Reset to Zero                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-6: Stimulus Program *(data_out)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program DATA_OUT"

! Set addressing mode and setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   setspace space (getspace space "memory", size "word")
   reset device devname
   sync device devname, mode "pod"
   sync device "/pod", mode "data"

! Present stimulus to UUT.

   arm device devname           ! Start response capture.
      rampdata addr 0, data 0, mask $FF
      rampdata addr 0, data 0, mask $FF00
   readout device devname       ! End response capture.

   podsetup 'enable ~ready' "on"
end program
```

Figure 4-6: Stimulus Program *(data_out) - continued*

```
STIMULUS PROGRAM NAME: DATA_OUT
DESCRIPTION:                                          SIZE:              982 BYTES

                  ------------------- Response Data --------------------
    Node          Learned                Async Clk Counter                    Priority
Signal Src         With          SIG      LVL  LVL  Mode      Counter Range     Pin

U3-11             PROBE         AA61           1  0 TRANS
U3-11             I/O MODULE    AA61           1  0 TRANS
U3-12             PROBE         99DF           1  0 TRANS
U3-12             I/O MODULE    99DF           1  0 TRANS
U3-13             PROBE         8793           1  0 TRANS
U3-13             I/O MODULE    8793           1  0 TRANS
U3-14             PROBE         E618           1  0 TRANS
U3-14             I/O MODULE    E618           1  0 TRANS
U3-15             PROBE         8793           1  0 TRANS
U3-15             I/O MODULE    F513           1  0 TRANS
U3-16             PROBE         4FFB           1  0 TRANS
U3-16             I/O MODULE    4FFB           1  0 TRANS
U3-17             PROBE         3600           1  0 TRANS
U3-17             I/O MODULE    3600           1  0 TRANS
U3-18             PROBE         B259           1  0 TRANS
U3-18             I/O MODULE    B259           1  0 TRANS
U23-11            I/O MODULE    96EC           1  0 TRANS
U23-12            I/O MODULE    725B           1  0 TRANS
U23-13            I/O MODULE    E5ED           1  0 TRANS
U23-14            I/O MODULE    5BE0           1  0 TRANS
U23-15            I/O MODULE    7E25           1  0 TRANS
U23-16            I/O MODULE    85EA           1  0 TRANS
U23-17            I/O MODULE    77C7           1  0 TRANS
U23-18            I/O MODULE    6EBE           1  0 TRANS
U14-51            PROBE         6EBE           1  0 TRANS
U14-51            I/O MODULE    6EBE           1  0 TRANS
U14-49            PROBE         77C7           1  0 TRANS
U14-49            I/O MODULE    77C7           1  0 TRANS
U14-47            PROBE         85EA           1  0 TRANS
U14-47            I/O MODULE    85EA           1  0 TRANS
U14-45            PROBE         7E25           1  0 TRANS
U14-45            I/O MODULE    7E25           1  0 TRANS
U14-43            PROBE         5BE0           1  0 TRANS
U14-43            I/O MODULE    5BE0           1  0 TRANS
U14-41            PROBE         E5ED           1  0 TRANS
U14-41            I/O MODULE    E5ED           1  0 TRANS
U14-39            PROBE         725B           1  0 TRANS
U14-39            I/O MODULE    725B           1  0 TRANS
U14-37            PROBE         96EC           1  0 TRANS
U14-37            I/O MODULE    96EC           1  0 TRANS
U14-50            PROBE         B259           1  0 TRANS
U14-50            I/O MODULE    B259           1  0 TRANS
U14-48            PROBE         3600           1  0 TRANS
```

(continued on the next page)

Figure 4-7: Response File *(data_out)*

```
U14-48      I/O MODULE  3600        1 0 TRANS
U14-46      PROBE       4FFB        1 0 TRANS
U14-46      I/O MODULE  4FFB        1 0 TRANS
U14-44      PROBE       F513        1 0 TRANS
U14-44      I/O MODULE  F513        1 0 TRANS
U14-42      PROBE       E618        1 0 TRANS
U14-42      I/O MODULE  E618        1 0 TRANS
U14-40      PROBE       8793        1 0 TRANS
U14-40      I/O MODULE  8793        1 0 TRANS
U14-38      PROBE       99DF        1 0 TRANS
U14-38      I/O MODULE  99DF        1 0 TRANS
U14-36      PROBE       AA61        1 0 TRANS
U14-36      I/O MODULE  AA61        1 0 TRANS
```

Figure 4-7: Response File *(data_out) - continued*

```
program ctrl_out1

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM for bus controller, U15 & uP ctrl lines.                    !
!                                                                              !
! Stimulus programs and response files are used by GFI to backtrace            !
! from a failing node.  The stimulus program must create repeatable UUT        !
! activity and the response file contains the known-good responses for         !
! the outputs in the UUT that are stimulated by the stimulus program.          !
!                                                                              !
! This stimulus program is one of the programs which creates activity          !
! in the kernel area of the UUT.  These programs create activity with          !
! or without the ready circuit working properly.  Because of this, all         !
! the stimulus programs in the kernel area must disable the READY input        !
! to the pod, then perform the stimulus, then re-enable the READY input        !
! to the pod.  The 80286 microprocessor has a separate bus controller;         !
! for this reason, disabling ready and performing stimulus can get the         !
! bus controller out of synchronization with the pod.  Two fault               !
! handlers trap pod timeout conditions that indicate the bus controller        !
! is out of synchronization.  The recover() program is executed to             !
! resynchronize the bus controller and the pod.                                !
!                                                                              !
! TEST PROGRAMS CALLED:                                                        !
!    recover     ()                         The 80286 microprocessor has a!
!                                           bus controller that is totally!
!                                           separate from the pod.  In    !
!                                           some cases the pod can get out!
!                                           of sync with the bus control- !
!                                           ler.  The recover program     !
!                                           resynchronizes the pod and the!
!                                           bus controller.               !
!                                                                              !
! GRAPHICS PROGRAMS CALLED:                                                    !
!    (none)                                                                    !
!                                                                              !
! Global Variables Modified:                                                   !
!    recover_times                          Reset to Zero                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
```

(continued on the next page)

Figure 4-8: Stimulus Program *(ctrl_out1)*

```
handle pod_timeout_recovered
   recover()
end handle
handle pod_timeout_no_clk
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
       devname = gfi device
   else
       devname = "/mod1"
   end if
   print "Stimulus Program CTRL_OUT1"

! Set addressing mode and setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   io_byte = getspace space "i/o", size "byte"
   mem_word = getspace space "memory", size "word"
   reset device devname
   sync device devname, mode "pod"
   sync device "/pod", mode "addr"
   old_cal = getoffset device devname
   setoffset device devname, offset (1000000 - 42)

! Present stimulus to UUT.

   arm device devname          ! Start response capture.
      setspace (mem_word)
      rampaddr addr $E0000, mask $1E
      rampdata addr $50000, data 0, mask $F
      setspace (io_byte)
      rampaddr addr 0, mask $5F00
      rampdata addr $2000, data 0, mask $F
   readout device devname      ! End response capture.

   setoffset device devname, offset old_cal
   podsetup 'enable ~ready' "on"
end program
```

Figure 4-8: Stimulus Program *(ctrl_out1) - continued*

```
STIMULUS PROGRAM NAME: CTRL_OUT1
DESCRIPTION:                                        SIZE:          267 BYTES

                  ------------------- Response Data --------------------
   Node            Learned                Async Clk Counter                    Priority
Signal Src          With          SIG       LVL  LVL  Mode     Counter Range     Pin

U14-5          PROBE          5632      1  0        TRANS
U14-5          I/O MODULE     5632      1  0        TRANS
U14-4          PROBE          ECCF      1  0        TRANS
U14-4          I/O MODULE     ECCF      1  0        TRANS
U14-66         PROBE          B70D      1  0        TRANS
U14-66         I/O MODULE     B70D      1  0        TRANS
U14-67         PROBE          0DF0      1  0        TRANS
U14-67         I/O MODULE     0DF0      1  0        TRANS
U45-8          I/O MODULE     92FB      1  0        TRANS
U15-16         I/O MODULE     2BE5      1  0        TRANS
U57-8          I/O MODULE     9118      1  0        TRANS
U22-5          I/O MODULE     B70D      1  0        TRANS
U22-6          I/O MODULE     0DF0      1  0        TRANS
```

Figure 4-9: Response File *(ctrl_out1)*

Summary of Complete Solution for Microprocessor Bus 4.1.7.

The entire set of programs and files needed to test and GFI troubleshoot the Microprocessor Bus functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Microprocessor Bus)

Programs (PROGRAM):

TEST_BUS	Functional test	Section 4.1.5
ADDR_OUT	Stimulus Program	Figure 4-4
DATA_OUT	Stimulus Program	Figure 4-6
CTRL_OUT1	Stimulus Program	Figure 4-8
LEVELS	Stimulus Program	Figure 4-92

Stimulus Program Responses (RESPONSE):

ADDR_OUT	Figure 4-5
DATA_OUT	Figure 4-7
CTRL_OUT1	Figure 4-9
LEVELS	Figure 4-93

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

(This page is intentionally blank.)

ROM FUNCTIONAL BLOCK 4.2.

Introduction to ROM 4.2.1.

The typical ROM block consists of the ROMs, an address path from the microprocessor to the ROMs, a data path from the ROMs to the microprocessor, and a ROM-select scheme. There are often hardware buffers separating the address and data paths from the microprocessor and ROMs; your UUT may or may not include these buffers. A simplified diagram of a typical ROM functional block is shown in Figure 4-10.

Figure 4-10 shows the microprocessor's Read/Write strobe as an input to the ROM-select circuitry. Many UUTs use the Read/Write strobe to make sure the ROM is selected only during Read cycles. This prevents potential data-bus contention between the ROM and the microprocessor during erroneous Write cycles to the ROM's address space.

Considerations for Testing and Troubleshooting 4.2.2.

Testing ROM

To test ROM thoroughly, every data bit read from the ROM (*i.e.*, every cell in the ROM) must be verified. Of course, you could compare the contents of every location with known-good contents, but this would be slow and would require that the 9100A/9105A store the known-good contents of all ROM chips. In practice, it is easier and faster to read every ROM address, compress the data into a CRC signature, and compare this signature with the signature from a known-good UUT.

The 9100A/9105A's built-in ROM test performs the operation described above. The test is first used to capture the signature

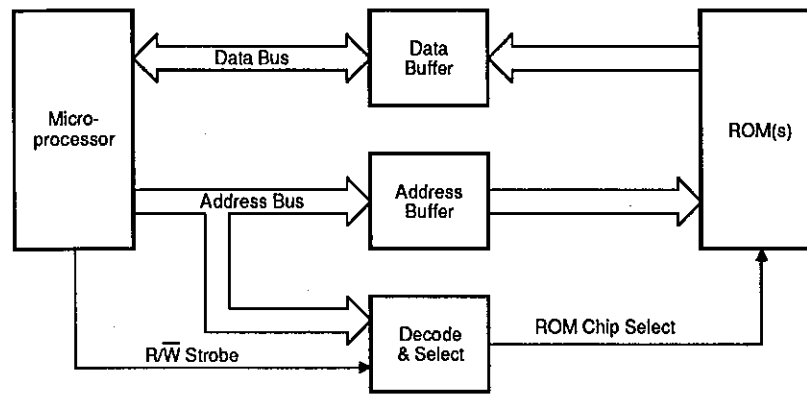


Figure 4-10: Typical ROM Block

response of a known-good UUT. Then, the test can be performed on a suspect UUT.

Refer to Section 6.2.3 of the *Technical User's Manual* for more information about the built-in ROM test.

ROM-Test Diagnostic Messages and Troubleshooting Techniques

If the built-in ROM test finds a fault, one of several diagnostic messages will be displayed. Figure 4-11 summarizes the types of conditions reported, with example messages. Here are some details about the various types of messages:

Incorrect Signature

This means that the ROM test could not identify the data or address lines at fault. It may indicate that the ROM chip itself is bad or that the wrong ROM chip is inserted. However, it could also indicate faulty ROM-select circuitry, especially if the circuitry allows ROM to be selected over only part of the proper address range. This type of fault would allow the test to read enough addresses to generate a signature, albeit an incorrect one. Here are some troubleshooting tips for this situation:

- Check that the correct ROM chip is plugged in.
- Perform the test on a known-good UUT with an I/O module clipped over the ROM chip. Write down the signatures of the individual lines from the I/O module.
- Perform the test on the suspect UUT, again with the I/O module clipped over the ROM chip.
- Compare the signatures for the individual lines. Trace any faulty inputs back toward the microprocessor, giving priority to tracing faults in chip-select lines and then in address lines.

Signal Group	*Fault*	*Example Message*
ROM Chip	bad data cells	read incorrect sig XXXX expected YYYY
ROM-Select Lines	open or stuck	read incorrect sig XXXX expected YYYY
		all ROM data bits stuck low
		all ROM data bits stuck high
Data Lines	open or stuck	data line <name> stuck high
		data line <name>stuck low
	tied	data line <name> tied to data line <name>
Address Lines	open or stuck	address line <name>stuck
	tied	address line <name>tied to address line <name>
Undetermined Fault		read incorrect sig XXXX expected YYYY

Figure 4-11: Conditions Reported by ROM Test

All Data Bits Stuck High or Low

This means that the ROM test found all ones or all zeroes on every data line throughout the test. Most probably, it means that the ROM chip is not being properly selected, that the ROM chip is missing (or unprogrammed), or that an intervening bus buffer is faulty.

To troubleshoot these faults, first check that the ROM chip is present and that it is the right part. If so, you can then trace the ROM-select path back to the microprocessor. Use a 9100A/9105A read operation on the address at which the failure occurred as a stimulus for the probe or I/O module. If the ROM-select path is good, verify that the address and data buffers are good.

Data or Address Line Stuck High, Stuck Low, or Tied

When an individual address or data line is at fault, use the probe to trace from the ROM socket back to the microprocessor and compare each node response with the known-good response.

If the faulty line is an address line, synchronize the probe to address and stimulate the line with the STIM key using the TOGGLE ADDR command on the operator's keypad. Use the LOOP key while probing to verify both low and high levels at each point on the address line until the fault is isolated.

If the faulty line is a data line, synchronize the probe to data, run a ROM test and press the LOOP key to repeat the ROM test while probing. Again, look for both low and high levels until the fault is isolated.

Additional Considerations

Here are some additional suggestions to consider when testing and troubleshooting ROM:

- **Multiple ROM Chips:** If you have more than one ROM chip on your UUT, test each chip separately. This will speed the troubleshooting process if a fault is found.

  If there is more than one ROM chip on the same data bus (or, in systems wider than 8 bits, on the same portion of the data bus) be careful that an erroneously enabled output buffer of one ROM is not corrupting the test results for another ROM. For example, consider an 8-bit microprocessor system with two ROM chips, A and B, in which chip A's output-enable input pin is tied low (a fault). Chip A will pass its ROM test, because the data in the ROM can still be read with the output-enable line tied low. ROM chip B, however, will fail its test with an incorrect-signature fault, even though there are no faults directly associated with chip B. When chip B is read by the test, the fault on chip A causes both ROMs to contend for the data bus, resulting in an incorrect signature. See the microprocessor bus functional block for suggestions on how to check for bus contention.

- **Unprogrammed ROM:** Be sure that the ROM being tested has been programmed. An unprogrammed ROM may result in an "all ROM data bits stuck high" or an "all ROM data bits stuck low" message during a ROM test.

- **Data Tied to Address:** If a ROM test results in a bad signature, it is a good idea to make sure that a data line is not tied to an address line. You can do so by clipping an I/O module to the ROM chip that produced the incorrect signature.

  If address line or data line failures are identified by a ROM test but not by a BUS test, the fault is on the ROM side of the address or data buffers.

- **Proper Sync Mode:** Generally, the data sync mode should be used to trace back faults in the ROM-select path, even though the ROM-select signal may be created from address lines. This is because the ROM-select signal should normally be asserted at the time the microprocessor reads in data from the ROM. This is also normally the situation for probing the address signals at the ROM socket.

ROM Example 4.2.3.

The operating system code for the Demo/Trainer UUT is stored in four 32K x 8 EPROMs, U27, U28, U29, and U30, shown in Figure 4-3. Since a 16-bit system is used, ROM is organized as 64K x 16 bits. The ROM0 bank covers the even addresses E0000 through EFFFE and is contained in U29 and U30. The ROM1 bank covers the even addresses F0000 through FFFFE and is contained in U27 and U28. Both banks can only be accessed in 16-bit mode. IA01 is connected to A0 on the ROMs, and the least significant address bit, IA00, is not connected to ROM. IA00 is always low in word accesses. A20-A23 are not used. At reset, 80286 code execution begins at the reset address (FFFFF0). ROM accesses do not require wait states.

Keystroke Functional Test 4.2.4.

Use the ROM TEST key to enter the following commands, and compare the measured signature with the response table in Figure 4-12.

```
GET SIG ROM REF U29 ADDR E0000 UPTO EFFFE ...
... DATA MASK FF ADDR STEP 2
... (ADDR OPTION: MEMORY WORD)
```

The measured signature (shown on the operator's display) should be 8E6E.

```
GET SIG ROM REF U30 ADDR E0000 UPTO EFFFE ...
... DATA FF00 ADDRSTEP 2
... (ADDR OPTION: MEMORY WORD)
```

The measured signature (shown on the operator's display) should be F387.

```
GET SIG ROM REF U27 ADDR F0000 UPTO FFFFE ...
... DATA FF ADDRSTEP 2
... (ADDR OPTION: MEMORY WORD)
```

The measured signature (shown on the operator's display) should be F387.

```
GET SIG ROM REF U28 ADDR F0000 UPTO FFFFE ...
... DATA FF00 ADDRSTEP 2
... (ADDR OPTION: MEMORY WORD)
```

The measured signature (shown on the operator's display) should be 8E6E.

(This page is intentionally blank.)

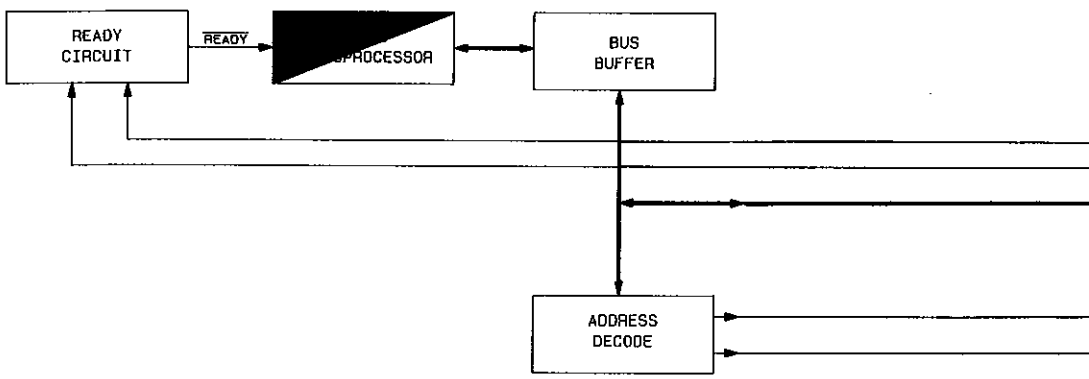
Keystroke Functional Test

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	POD TEST ACCESS SOCKET

RESPONSE

ROM CHIP	ROM SIGNATURE
U29 U30 U27 U28	8E6E F387 F387 8E6E



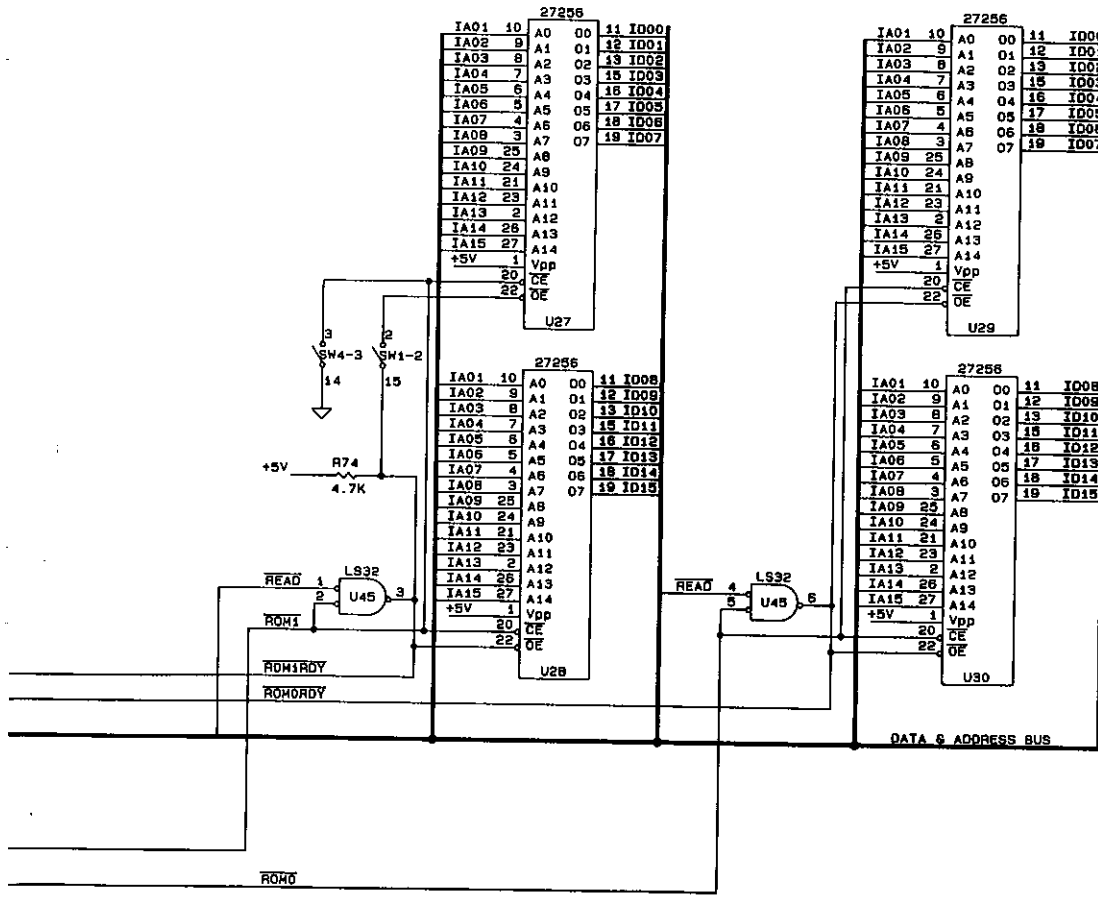


Figure 4-12: ROM Functional Test

Programmed Functional Test 4.2.5.

The *test_rom* program is the programmed functional test for the ROM functional block. It uses the *testromful* command to test the ROMs. This command will generate one of seven built-in fault conditions if *testromful* fails. The *test_rom* program then handles all seven built-in fault conditions and categorizes them into one of two new fault conditions called *rom_comp* for a component failure or *rom_address* for an address failure. The seven built-in *testramfull* faults are redirected as follows:

New Fault Condition	*Built-in Fault Condition*
rom_comp	rom_sig_incorrect rom_data_high_tied_all rom_data_low_tied_all rom_data_fault rom_data_data_tied
rom_address	rom_addr_addr_tied rom_addr_fault

The new fault condition *rom_comp* uses the *gfi test* command to clip the I/O module onto the ROMs and to test all inputs and outputs of a ROM. If a failure is detected, the test passes control to GFI. GFI backtraces to find the circuit problem that is causing the failure.

The new fault condition *rom_address* checks the address bus, and if a failure is detected control is passed to GFI. GFI then backtraces to the circuit problem which is causing the failure.

```
program test_rom

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the ROM functional block.                               !
!                                                                            !
! This program tests the ROM functional block of the Demo/Trainer.  The      !
! TL/1 testromfull command is used to test the ROMs.  If the ROMs are        !
! found to be faulty, then one of seven built-in fault conditions is         !
! generated.                                                                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Setup.

   podsetup 'enable ~ready' "on"
   podsetup 'report forcing' "on"
   setspace space (getspace space "memory", size "word")

! Main part of Test.

   testromfull addr $F0000, upto $FFFFE, addrstep 2, sig $156F
   testromfull addr $E0000, upto $EFFFE, addrstep 2, sig $B61E

end program
```

Stimulus Programs and Responses 4.2.6.

Figure 4-13 is the stimulus program planning diagram for the ROM functional block. The outputs in the ROM functional block are the outputs of U45 and the outputs of the ROM chips onto the data bus.

The stimulus programs to exercise these outputs are *rom0_data* (which reads data from U29 and U30), *rom1_data* (which reads data from U27 and U28), and *decode* (which accesses each decoded address space in the Demo/Trainer UUT).

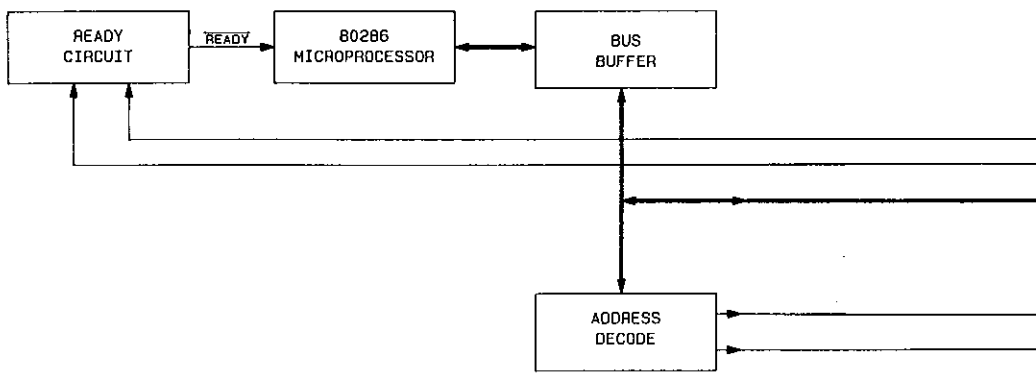
One of the rules for stimulus programs is that when dealing with a data bus, every component that is decoded separately to output onto the data bus must have a separate stimulus program to read data from that component. For this reason, two stimulus programs are required: *rom0_data* and *rom1_data*.

(This page is intentionally blank.)

Stimulus Program Planning

PROGRAM: ROM0_DATA
READS FIRST 2K OF DATA FROM ROMs U29 AND U30
MEASUREMENT AT: U29-11,12,13,15,16,17,18,19 U30-11,12,13,15,16,17,18,19

PROGRAM: ROM1_DATA
READS FIRST 2K OF DATA FROM ROMs U27 AND U28
MEASUREMENT AT: U27-11,12,13,15,16,17,18,19 U28-11,12,13,15,16,17,18,19



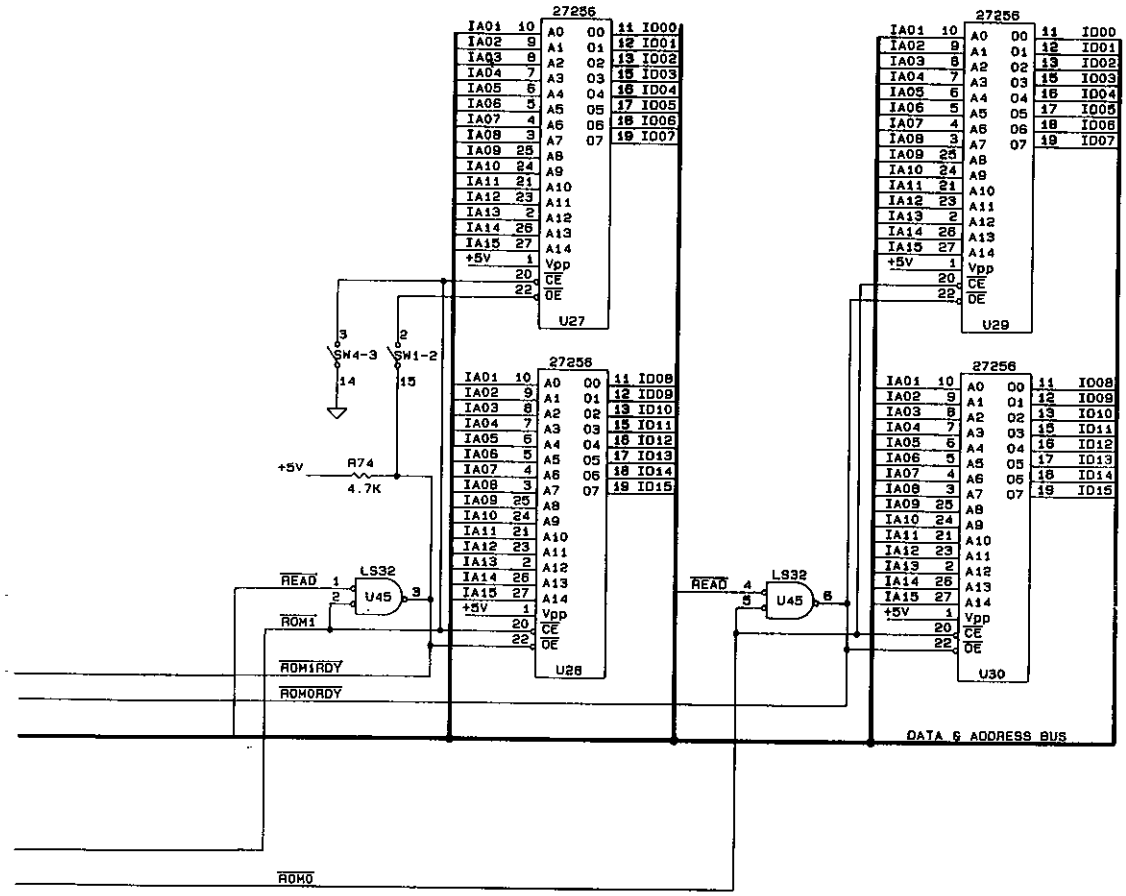


Figure 4-13: ROM Stimulus Program Planning

```
program rom0_data

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to exercise data out of ROMs U29 and U30.              !
!                                                                         !
! Stimulus programs and response files are used by GFI to backtrace       !
! from a failing node.  The stimulus program must create repeatable UUT   !
! activity and the response file contains the known-good responses for    !
! the outputs in the UUT that are stimulated by the stimulus program.     !
!                                                                         !
! This stimulus program is one of the programs which creates activity     !
! in the kernel area of the UUT.  These programs create activity with     !
! or without the ready circuit working properly.  Because of this, all    !
! the stimulus programs in the kernel area must disable the READY input   !
! to the pod, then perform the stimulus, then re-enable the READY input   !
! to the pod.  The 80286 microprocessor has a separate bus controller;    !
! for this reason, disabling ready and performing stimulus can get the    !
! bus controller out of synchronization with the pod.  Two fault          !
! handlers trap pod timeout conditions that indicate the bus controller   !
! is out of synchronization.  The recover() program is executed to        !
! resynchronize the bus controller and the pod.                           !
!                                                                         !
! TEST PROGRAMS CALLED:                                                   !
!    recover     ()                         The 80286 microprocessor has a!
!                                           bus controller that is totally!
!                                           separate from the pod.  In    !
!                                           some cases the pod can get out!
!                                           of sync with the bus control- !
!                                           ler.  The recover program     !
!                                           resynchronizes the pod and the!
!                                           bus controller.               !
!                                                                         !
! GRAPHICS PROGRAMS CALLED:                                               !
!    (none)                                                               !
!                                                                         !
! Global Variables Modified:                                              !
!    recover_times                          Reset to Zero                 !
!    devname                                Measurement device            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-14: Stimulus Program *(rom0_data)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI user select which I/O module to use

  if (gfi control) = "yes" then
    devname = gfi device
  else
    devname = "/mod1"
  end if
   print "Stimulus Program ROM0_DATA"

! Set desired measurement modes

  setspace space (getspace space "memory", size "word")
  reset device devname
  sync device devname, mode "pod"
  sync device "/pod", mode "data"

! Present stimulus to the UUT

  arm device devname                   ! Start response capture.
     rampaddr addr $E0000, mask $1FE
  readout device devname               ! End response capture

end rom0_data
```

Figure 4-14: Stimulus Program *(rom0_data) - continued*

STIMULUS PROGRAM NAME: ROM0_DATA
DESCRIPTION: SIZE: 454 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U29-11	PROBE	45DD		1	0 TRANS		
U29-11	I/O MODULE	45DD		1	0 TRANS		
U29-12	PROBE	CF83		1	0 TRANS		
U29-12	I/O MODULE	CF83		1	0 TRANS		
U29-13	PROBE	BD79		1	0 TRANS		
U29-13	I/O MODULE	BD79		1	0 TRANS		
U29-15	PROBE	8A76		1	0 TRANS		
U29-15	I/O MODULE	8A76		1	0 TRANS		
U29-16	PROBE	66F3		1	0 TRANS		
U29-16	I/O MODULE	66F3		1	0 TRANS		
U29-17	PROBE	FAB5		1	0 TRANS		
U29-17	I/O MODULE	FAB5		1	0 TRANS		
U29-18	PROBE	534E		1	0 TRANS		
U29-18	I/O MODULE	534E		1	0 TRANS		
U29-19	PROBE	8D0A		1	0 TRANS		
U29-19	I/O MODULE	8D0A		1	0 TRANS		
U30-11	I/O MODULE	73E9		1	0 TRANS		
U30-12	I/O MODULE	AC84		1	0 TRANS		
U30-13	I/O MODULE	50BB		1	0 TRANS		
U30-15	I/O MODULE	5B3B		1	0 TRANS		
U30-16	I/O MODULE	06EF		1	0 TRANS		
U30-17	I/O MODULE	00A0		1	0 TRANS		
U30-18	I/O MODULE	6BF0		1	0 TRANS		
U30-19	I/O MODULE	52EE		1	0 TRANS		

Figure 4-15: Response File *(rom0_data)*

```
program rom1_data

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to exercise data out of ROMs U29 and U30.              !
!                                                                         !
! Stimulus programs and response files are used by GFI to back-trace      !
! from a failing node.  The stimulus program must create repeatable UUT   !
! activity and the response file contains the known-good responses for    !
! the outputs in the UUT that are stimulated by the stimulus program.     !
!                                                                         !
! This stimulus program is one of the programs which creates activity     !
! in the kernel area of the UUT.  These programs create activity with     !
! or without the ready circuit working properly.  Because of this, all    !
! the stimulus programs in the kernel area must disable the READY input   !
! to the pod, then perform the stimulus, then re-enable the READY input   !
! to the pod.  The 80286 microprocessor has a separate bus controller;    !
! for this reason, disabling ready and performing stimulus can get the    !
! bus controller out of synchronization with the pod.  Two fault          !
! handlers trap pod timeout conditions that indicate the bus controller   !
! is out of synchronization.  The recover() program is executed to        !
! resynchronize the bus controller and the pod.                           !
!                                                                         !
! TEST PROGRAMS CALLED:                                                   !
!    recover     ()                          The 80286 microprocessor has a!
!                                            bus controller that is totaly !
!                                            separate from the pod.  In    !
!                                            some cases the pod can get out!
!                                            of sync with the bus control- !
!                                            ler.  The recover program     !
!                                            resynchronizes the pod and the!
!                                            bus controller.               !
!                                                                         !
! GRAPHICS PROGRAMS CALLED:                                               !
!    (none)                                                               !
!                                                                         !
! Global Variables Modified:                                              !
!    recover_times                           Reset to Zero                !
!    devname                                 Measurement device           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-16: Stimulus Program *(rom1_data)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI user select which I/O module to use

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program ROM1_DATA"

! Set desired measurement modes

   setspace space (getspace space "memory", size "word")
   reset device devname
   sync device devname, mode "pod"
   sync device "/pod", mode "data"

! Present stimulus to the UUT

   arm device devname              ! Start response capture.
      rampaddr addr $F0000, mask $1FE
   readout device devname          ! End response capture

end program
```

Figure 4-16: Stimulus Program *(rom1_data) - continued*

STIMULUS PROGRAM NAME: ROM1_DATA
DESCRIPTION: SIZE: 982 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U27-11	PROBE	73E9		1	0 TRANS		
U27-11	I/O MODULE	73E9		1	0 TRANS		
U27-12	PROBE	AC84		1	0 TRANS		
U27-12	I/O MODULE	AC84		1	0 TRANS		
U27-13	PROBE	50BB		1	0 TRANS		
U27-13	I/O MODULE	50BB		1	0 TRANS		
U27-15	PROBE	5B3B		1	0 TRANS		
U27-15	I/O MODULE	5B3B		1	0 TRANS		
U27-16	PROBE	06EF		1	0 TRANS		
U27-16	I/O MODULE	06EF		1	0 TRANS		
U27-17	PROBE	00A0		1	0 TRANS		
U27-17	I/O MODULE	00A0		1	0 TRANS		
U27-18	PROBE	6BF0		1	0 TRANS		
U27-18	I/O MODULE	6BF0		1	0 TRANS		
U27-19	PROBE	52EE		1	0 TRANS		
U27-19	I/O MODULE	52EE		1	0 TRANS		
U28-11	I/O MODULE	45DD		1	0 TRANS		
U28-12	I/O MODULE	CF83		1	0 TRANS		
U28-13	I/O MODULE	BD79		1	0 TRANS		
U28-15	I/O MODULE	8A76		1	0 TRANS		
U28-16	I/O MODULE	66F3		1	0 TRANS		
U28-17	I/O MODULE	FAB5		1	0 TRANS		
U28-18	I/O MODULE	534E		1	0 TRANS		
U28-19	I/O MODULE	8D0A		1	0 TRANS		
U23-2	PROBE	52EE		1	0 TRANS		
U23-2	I/O MODULE	52EE		1	0 TRANS		

(continued on the next page)

Figure 4-17: Response File *(rom1_data)*

```
U23-3       PROBE       6BF0        1 0 TRANS
U23-3       I/O MODULE  6BF0        1 0 TRANS
U23-4       PROBE       00A0        1 0 TRANS
U23-4       I/O MODULE  00A0        1 0 TRANS
U23-5       PROBE       06EF        1 0 TRANS
U23-5       I/O MODULE  06EF        1 0 TRANS
U23-6       PROBE       5B3B        1 0 TRANS
U23-6       I/O MODULE  5B3B        1 0 TRANS
U23-7       PROBE       50BB        1 0 TRANS
U23-7       I/O MODULE  50BB        1 0 TRANS
U23-8       PROBE       AC84        1 0 TRANS
U23-8       I/O MODULE  AC84        1 0 TRANS
U23-9       PROBE       73E9        1 0 TRANS
U23-9       I/O MODULE  73E9        1 0 TRANS
U3-2        PROBE       8D0A        1 0 TRANS
U3-2        I/O MODULE  8D0A        1 0 TRANS
U3-3        PROBE       534E        1 0 TRANS
U3-3        I/O MODULE  534E        1 0 TRANS
U3-4        PROBE       FAB5        1 0 TRANS
U3-4        I/O MODULE  FAB5        1 0 TRANS
U3-5        PROBE       66F3        1 0 TRANS
U3-5        I/O MODULE  66F3        1 0 TRANS
U3-6        PROBE       8A76        1 0 TRANS
U3-6        I/O MODULE  8A76        1 0 TRANS
U3-7        PROBE       BD79        1 0 TRANS
U3-7        I/O MODULE  BD79        1 0 TRANS
U3-8        PROBE       CF83        1 0 TRANS
U3-8        I/O MODULE  CF83        1 0 TRANS
U3-9        PROBE       45DD        1 0 TRANS
U3-9        I/O MODULE  45DD        1 0 TRANS
```

Figure 4-17: Response File *(rom1_data) - continued*

Summary of Complete Solution for ROM 4.2.7.

The entire set of programs and files needed to test and GFI troubleshoot the ROM functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for ROM)

Programs (PROGRAM):		
TEST_ROM	Functional Test	Section 4.2.5
ROM0_DATA	Stimulus Program	Figure 4-14
ROM1_DATA	Stimulus Program	Figure 4-16
DECODE	Stimulus Program	Figure 4-108
Stimulus Program Responses (RESPONSE):		
ROM0_DATA		Figure 4-15
ROM1_DATA		Figure 4-17
DECODE		Figure 4-109
Node List (NODE):		
NODELIST		Appendix B
Text Files (TEXT):		
Reference Designator List (REF):		
REFLIST		Appendix A
Compiled Database (DATABASE):		
GFIDATA		Compiled by the 9100A

(This page is intentionally blank.)

RAM FUNCTIONAL BLOCK 4.3.

Introduction to RAM 4.3.1.

The typical RAM block consists of the RAM chips, an address path from the microprocessor to the RAMs, a bidirectional data path between the microprocessor and the RAMs, and RAM-select circuitry. There are often hardware buffers between the microprocessor and the RAM chips.

There are two basic types of RAM: static and dynamic. Static RAM chips are faster and require no refresh circuitry. They are also more expensive and take more room for a given memory size. Dynamic RAM chips use a capacitor for charge storage and therefore must be periodically refreshed to maintain data storage. However, dynamic RAM chips provide more memory for a given size chip.

A simplified diagram of a typical RAM functional block is shown in Figure 4-18.

Considerations for Testing and Troubleshooting 4.3.2.

Speed and accuracy are the most critical factors in RAM testing, and RAM tests are typically a compromise between these two factors. To further complicate the issue, different hardware configurations bring with them different failure mechanisms which may require specialized testing.

The built-in RAM tests offers a number of choices to better match the test to the testing needs. While the RAM FULL, RAM FAST and pod-dependent RAM QUICK tests directly address the speed and accuracy compromise, they are different from each other.

Section 5 of the *Technical User's Manual* describes the various RAM tests in detail.

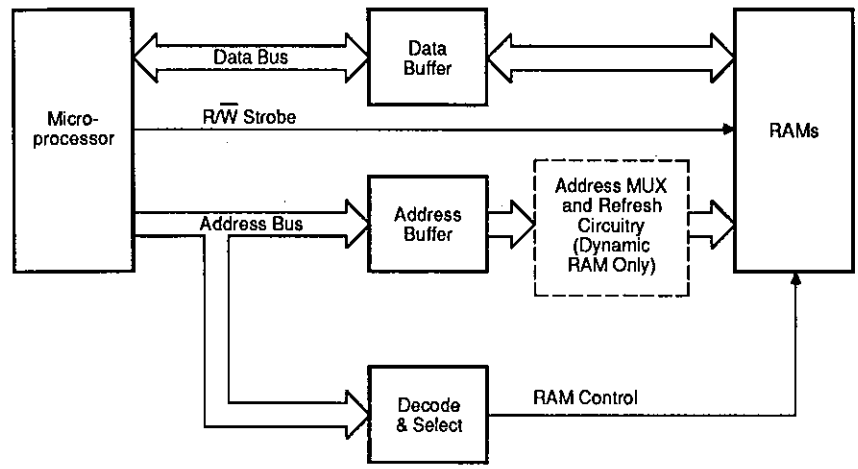


Figure 4-18: Typical RAM Block

Many types of faults can occur in RAM functional blocks. Address lines or data lines can be stuck or tied to other lines. Individual memory cells can be stuck low or high, or cells can be aliased (they respond to more than one address). Transition faults can exist (where a cell can change from one state to another, but not back again). Coupling faults can cause the contents of one cell to be disturbed when the contents of another cell is changed. If this coupling depends on the contents of several neighboring cells, the fault is called a pattern sensitive fault. Chip select address decoding logic can be faulty. Row or column decoders might not select when they should or they might select when they shouldn't. In dynamic memory, refresh logic can fail, causing cells to lose their contents.

Although failure mechanisms are different between dynamic and static RAM, both types of RAM may be functionally tested with exactly the same built-in RAM tests; only the delay parameter is of unique concern for dynamic RAM. The delay parameter provides a means of testing the refresh circuitry by specifying the number of milliseconds to wait for refresh-related faults to occur.

The first step in troubleshooting RAM is to run a built-in functional test. Besides confirming a RAM fault, the functional test often provides excellent clues for where to begin fault isolation. Figure 4-19 illustrates typical fault information provided by the RAM tests.

In general, the following procedure will work for troubleshooting any RAM faults discovered by the 9100A/9105A:

1. Create a combination of reads and writes to confirm the failure.
2. Synchronize the probe as needed.
3. Perform looping reads and writes while tracing with synchronized probe.

Fault Condition	*TEST RAM FAST*	*TEST RAM FULL*	
		coupling enabled	*coupling disabled*
Stuck cells	always found	always found	always found
Aliased cells	"	"	"
Stuck address lines	"	"	"
Stuck data lines	"	"	"
Shorted address lines	"	"	"
Multiple selection decoder	may be found	always found	always found
Dynamic coupling	"	"	"
Shorted data lines	may be found	always found	may be found
Aliasing between bits in same word	"	"	"
Pattern-sensitive faults	not found	not found	not found
Refresh problems	always found, if delay is sufficiently long and standby reads do not mask the problem.		

Figure 4-19: RAM Test Failure Information

RAM Example 4.3.3.

The Demo/Trainer UUT, Figure 4-20, uses 128K bytes of dynamic RAM, organized as 64k x 16 bits, and composed of sixteen 1-bit wide 4164 chips.

Keystroke Functional Test 4.3.4.

Use the RAM TEST key to enter the following command:

```
TEST RAM FAST ADDR 0 UPTO 1FFFE DATA MASK ...
... FFFF ADDR STEP 2 DELAY 250 SEED 0
... (ADDR OPTION: MEMORY WORD)
```

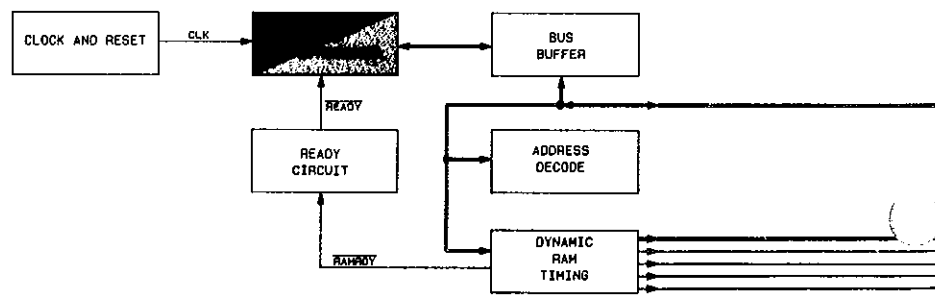
Keystroke Functional Test

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	POD TEST ACCESS SOCKET

RESPONSE

(BUILT-IN RESPONSE MEASUREMENT)



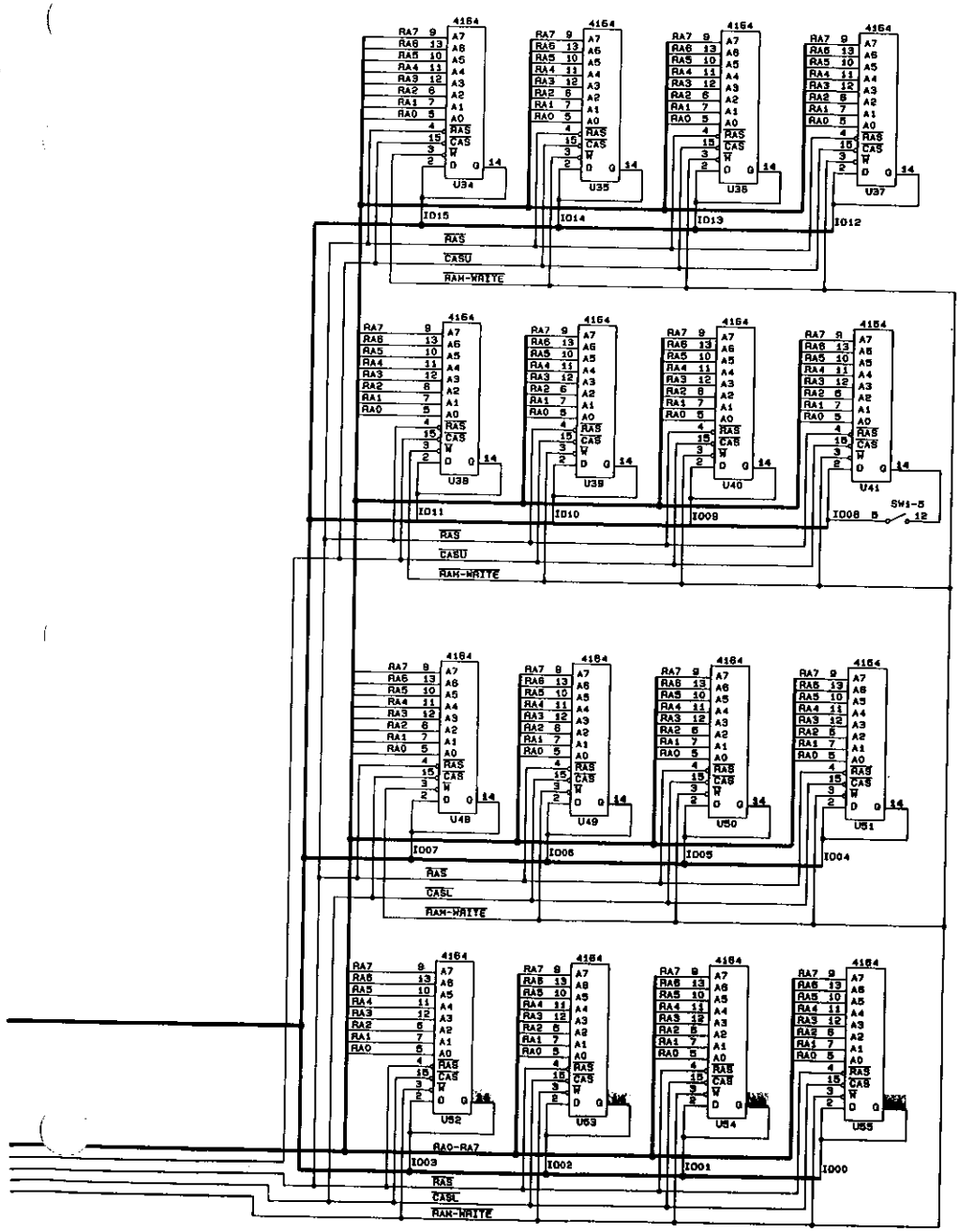


Figure 4-20: RAM Functional Test

Programmed Functional Test 4.3.5.

The *test_ram* program is the programmed functional test of the Dynamic RAM functional block. This program uses the *testramfast* command to test the RAM. This command will generate one of eleven different fault conditions if the *testramfast* fails. All eleven fault condition handlers pick up some parameters and redirect the fault condition to a new fault condition called *ram_component*. The fault condition handler for the *ram_component* fault condition accepts a parameter called *data_bits* that indicates which data bit positions are faulty.

The *ram_component* fault condition handler first checks the Ready circuit to make sure that a ready fault condition is not causing RAM failures. If the Ready circuit is good, one of the failing RAMs (as indicated by the *data_bits* parameter) is checked using the *gfi test* command. If a failure is found, GFI takes control and backtraces to the circuit fault causing the failure.

If the RAM component is good, the *ram_component* fault condition handler uses the *gfi test* command to check the data bus at the bus buffers. If a failure is detected, GFI begins backtracing from the bus buffers.

```
program test_ram

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the RAM functional block.                         !
!                                                                      !
! This program tests the RAM functional block of the Demo/Trainer. The !
! TL/1 testramfast command is used to test the RAMs. If the RAMs are   !
! found to be faulty, then one of twelve built-in fault conditions is  !
! generated.                                                           !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Setup

   podsetup 'enable ~ready' "on"
   podsetup 'report forcing' "on"
   setspace space (getspace space "memory", size "word")

! Main part of test

   testramfast addr 0, upto $1FFFE, delay 250, seed 1

end program
```

Stimulus Programs and Responses 4.3.6.

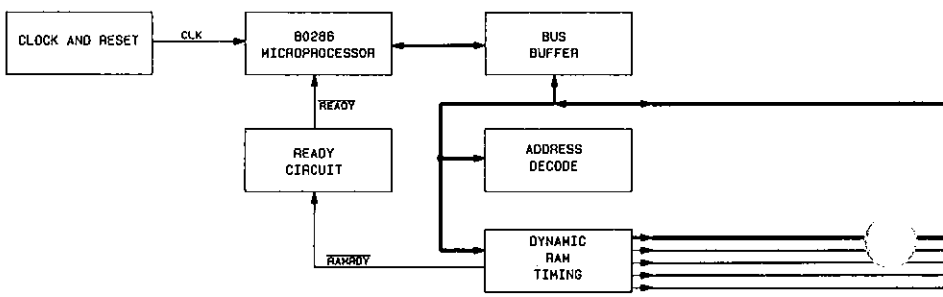
Figure 4-21 is the stimulus program planning diagram for the RAM functional block. There is one stimulus program and a matching response file for RAM. The stimulus program *ram_data* outputs data from RAM onto the data bus.

One rule for a stimulus program is that data should flow in only one direction during the measurement portion of the stimulus program. Although *ram_data* executes *ram_fill* in order to fill RAM with known data, *ram_fill* is executed before the measurement is started in the *ram_data* stimulus program. Therefore data will flow only in one direction during the measurement portion of *ram_data.*

Stimulus Program Planning

PROGRAM: RAM_DATA
EXECUTES RAM..FILL AND READS FROM THE FIRST 512 LOCATIONS OF RAM
MEASUREMENT AT:
U55-14 U51-14 U41-14 U37-14 U54-14 U50-14 U40-14 U36-14 U53-14 U49-14 U39-14 U35-14 U52-14 U48-14 U38-14 U34-14

INITIALIZATION PROGRAM: RAM_FILL
INITIALIZES RAM BY FILLING THE FIRST 512 LOCATIONS OF RAM WITH STANDARD DATA
MEASUREMENT AT:
(NONE)



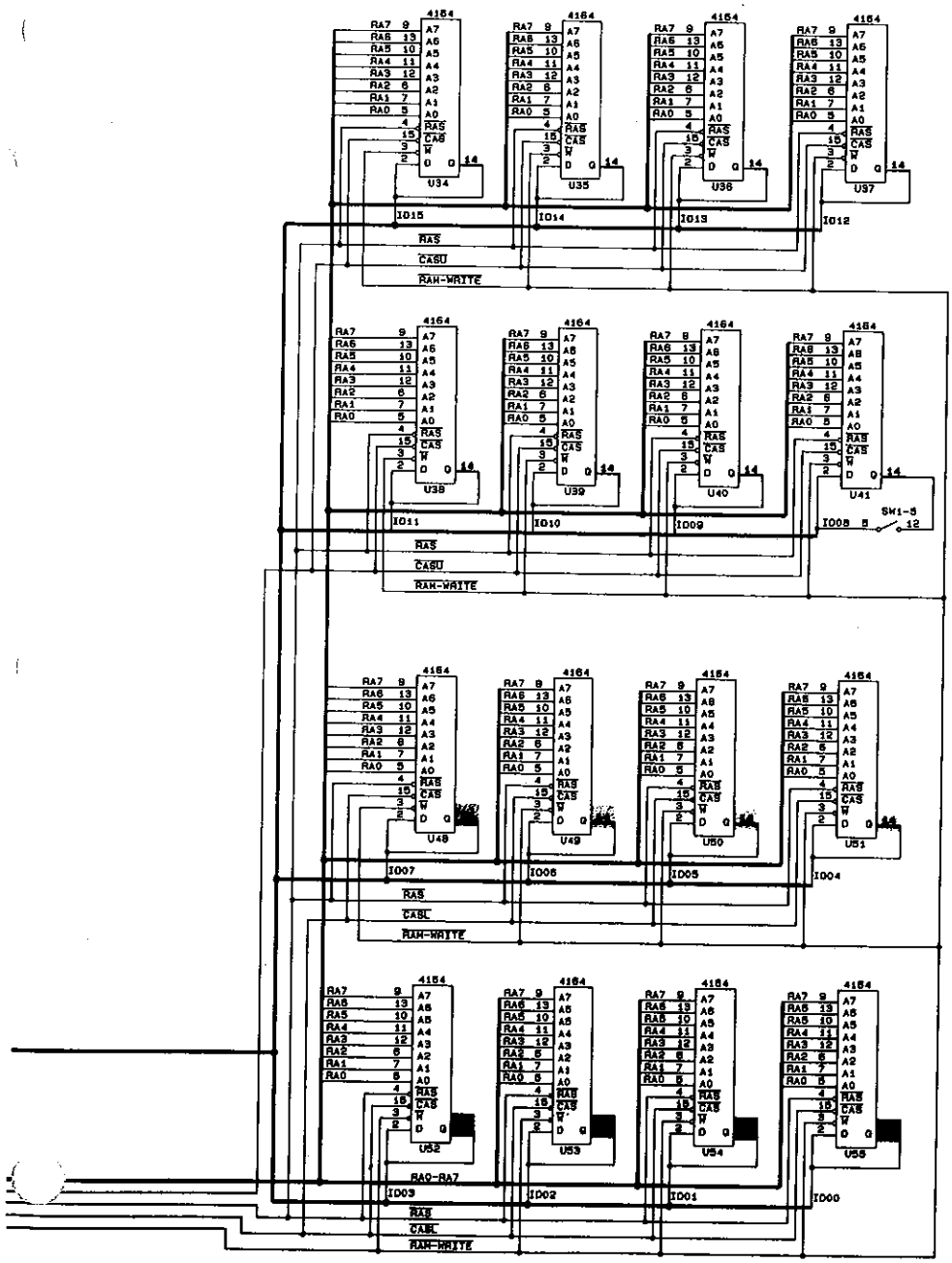


Figure 4-21: RAM Stimulus Program Planning

```
program ram_data

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to exercise data out of the dynamic RAM.                !
!                                                                          !
! Stimulus programs and response files are used by GFI to backtrace        !
! from a failing node.  The stimulus program must create repeatable UUT    !
! activity and the response file contains the known-good responses for     !
! the outputs in the UUT that are stimulated by the stimulus program.      !
!                                                                          !
! This stimulus program is one of the programs which creates activity      !
! in the kernel area of the UUT.  These programs create activity with      !
! or without the ready circuit working properly.  Because of this, all     !
! the stimulus programs in the kernel area must disable the READY input    !
! to the pod, then perform the stimulus, then re-enable the READY input    !
! to the pod.  The 80286 microprocessor has a separate bus controller;     !
! for this reason, disabling ready and performing stimulus can get the     !
! bus controller out of synchronization with the pod.  Two fault           !
! handlers trap pod timeout conditions that indicate the bus controller    !
! is out of synchronization.  The recover() program is executed to         !
! resynchronize the bus controller and the pod.                            !
!                                                                          !
! TEST PROGRAMS CALLED:                                                    !
!    dram_filll ()                        Initialize data in the RAM       !
!                                                                          !
!    recover     ()                       The 80286 microprocessor has a!
!                                         bus controller that is totaly !
!                                         separate from the pod.  In     !
!                                         some cases the pod can get out!
!                                         of sync with the bus control- !
!                                         ler.  The recover program     !
!                                         resynchronizes the pod and the!
!                                         bus controller.               !
!                                                                          !
! GRAPHICS PROGRAMS CALLED:                                                !
!    (none)                                                                !
!                                                                          !
! Local Variables Modified:                                                !
!    devname                              Measurement device               !
!                                                                          !
! Global Variables Modified:                                               !
!    recover_times                        Reset to Zero                    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-22: Stimulus Program *(ram_data)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI user select which I/O module to use

   if (gfi control) = "yes" then
     devname = gfi device
   else
     devname = "/mod1"
   end if
   print "Stimulus Program RAM_DATA"

! Set desired measurement modes

   reset device devname
   execute ram_fill()
   setspace space (getspace space "memory", size "word")
   sync device devname, mode "pod"
   sync device "/pod", mode "data"

! Present stimulus:  Read data out of RAM

   arm device devname                ! Start response capture.
      rampaddr addr 0, mask $1FE
   readout device devname            ! End response capture

end program
```

Figure 4-22: Stimulus Program *(ram_data) - continued*

STIMULUS PROGRAM: RAM_DATA
DESCRIPTION: SIZE: 454 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U34-14	I/O MODULE	95A1		1	0 TRANS		
U35-14	I/O MODULE	6F97		1	0 TRANS		
U36-14	I/O MODULE	7744		1	0 TRANS		
U37-14	I/O MODULE	5AE5		1	0 TRANS		
U38-14	I/O MODULE	A54D		1	0 TRANS		
U39-14	I/O MODULE	797B		1	0 TRANS		
U40-14	I/O MODULE	A5F7		1	0 TRANS		
U41-14	I/O MODULE	3BEF		1	0 TRANS		
U48-14	PROBE	C0A6		1	0 TRANS		
U48-14	I/O MODULE	C0A6		1	0 TRANS		
U49-14	PROBE	1338		1	0 TRANS		
U49-14	I/O MODULE	1338		1	0 TRANS		
U50-14	PROBE	66F9		1	0 TRANS		
U50-14	I/O MODULE	66F9		1	0 TRANS		
U51-14	PROBE	6CF8		1	0 TRANS		
U51-14	I/O MODULE	6CF8		1	0 TRANS		
U52-14	PROBE	BE05		1	0 TRANS		
U52-14	I/O MODULE	BE05		1	0 TRANS		
U53-14	PROBE	3C7C		1	0 TRANS		
U53-14	I/O MODULE	3C7C		1	0 TRANS		
U54-14	PROBE	70F3		1	0 TRANS		
U54-14	I/O MODULE	70F3		1	0 TRANS		
U55-14	PROBE	DACC		1	0 TRANS		
U55-14	I/O MODULE	DACC		1	0 TRANS		

Figure 4-23: Response File *(ram_data)*

```
program ram_fill

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! INITIALIZATION PROGRAM fills Dynamic RAM with a pattern of data.
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    (none)                                                             !
!                                                                       !
! GRAPHICS PROGRAMS CALLED:                                             !
!    (none)                                                             !
!                                                                       !
! Text Files Accessed:                                                  !
!    dram_filll                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   setspace space (getspace space "memory", size "word")
   writeblock file "dram_filll", format "motorola"

end program
```

Figure 4-24: Inititalization Program *(ram_fill)*

Summary of Complete Solution for RAM 4.3.7.

The entire set of programs and files needed to test and GFI troubleshoot the RAM functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for RAM)

Programs (PROGRAM):

TEST_RAM	Functional Test	Section 4.3.5
RAM_DATA	Stimulus Program	Figure 4-22
RAM_FILL	Initialization Program	Figure 4-24

Stimulus Program Responses (RESPONSE):

RAM_DATA	Figure 4-23

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

DRAM_FILL1	Initialization Data File

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

DYNAMIC RAM TIMING FUNCTIONAL BLOCK 4.4.

Introduction to Dynamic RAM Timing Circuits 4.4.1.

Unlike static RAM, dynamic RAM chips use a capacitor for charge storage and therefore must be periodically refreshed to maintain the data in memory. Refreshing does not require that data be re-written at memory locations; it requires only that every row be accessed within a certain time period (typically at least every 2 milliseconds). This is sufficient to restore the charge on the memory cells.

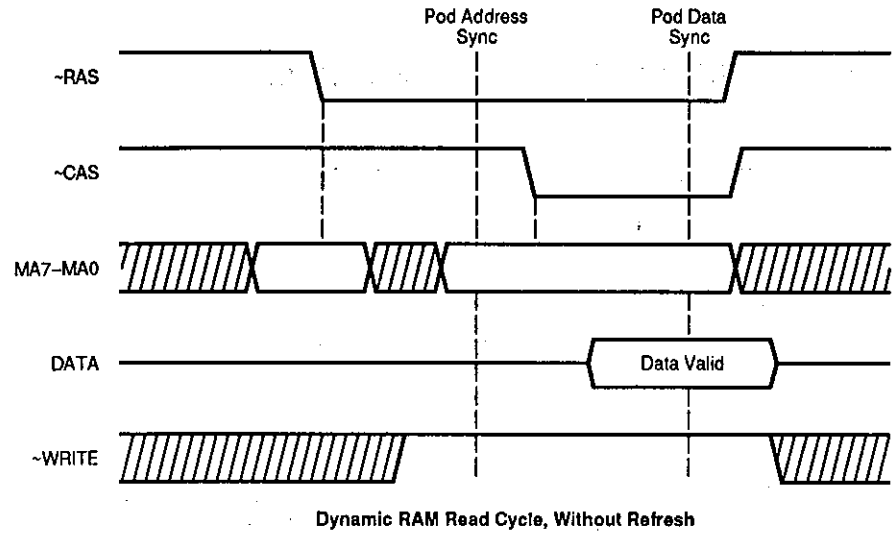
In addition, dynamic RAM uses multiplexed address signals. The row address is clocked into the internal decoder of the dynamic RAM chip with the falling edge of the Row Address Strobe (RAS), and the column address is clocked with the falling edge of the Column Address Strobe (CAS). Multiplexed addressing decreases the pin count and package size, but it also makes dynamic RAM more difficult to test and troubleshoot than static RAM.

Considerations for Testing and Troubleshooting 4.4.2.

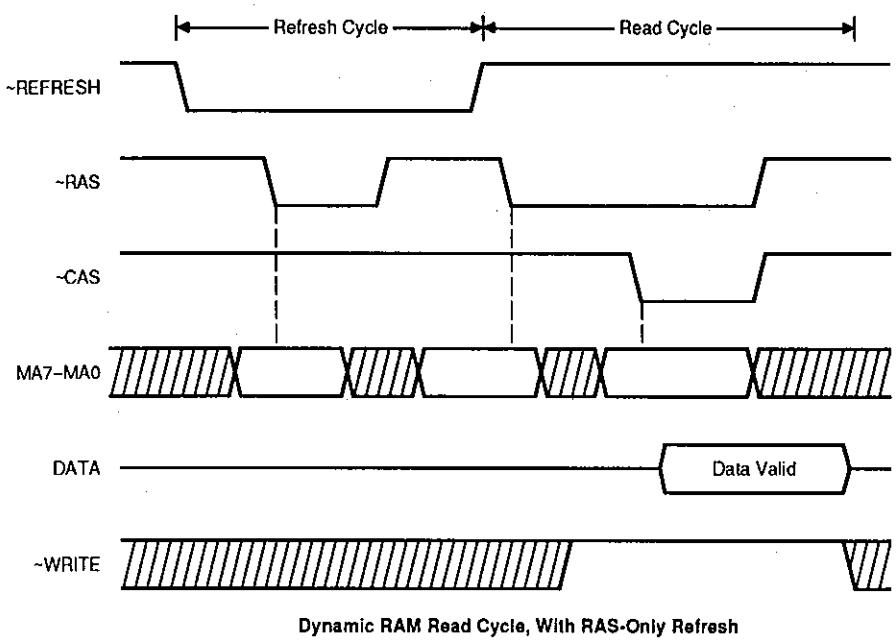
The thought process used to test and troubleshoot dynamic RAM is very similar to that used for static RAM, but the actual measurements for dynamic RAM are more difficult because of row and column strobing for multiplexed addresses, because of refreshing, and because there are more failure mechanisms.

Consider, for example, a dynamic RAM with 64K memory locations addressed by eight address inputs (MA7-MA0). A multiplexer allows the 16 address lines to be brought to the eight RAM address lines, using RAS to strobe the row address and CAS to strobe the column address. Typical timing for a read cycle of such a system is shown in Figure 4-25.

With static RAM, the microprocessor's address lines can be tested by making measurements using the probe or an I/O



Dynamic RAM Read Cycle, Without Refresh



Dynamic RAM Read Cycle, With RAS-Only Refresh

Figure 4-25: Dynamic RAM Read Cycles

module synchronized to the pod address while performing looping reads or writes. With dynamic RAM, however, the RAM's address inputs are multiplexed between row and column addresses. It is important to be able to separate row addressing from column addressing. To test dynamic RAM addressing requires the ability to control the timing of the clock strobe for a measurement. The 9100/9105A has this capability; under program control, it can adjust the timing for when the probe or I/O module actually clocks data. Using the *getoffset* and *setoffset* commands, you can create a program to measure the address line activity on the RAM chips at the RAS strobe (or at the CAS strobe). Typically, it makes sense to have two separate programs: one to measure activity for RAS address timing and one to measure activity for CAS address timing.

For the top example of Figure 4-25, the TL/1 *setoffset* and *getoffset* commands are used to adjust the sync timing from Pod Data Sync (or Pod Address Sync) to the RAS and CAS positions. One program could be used to measure at RAS time and another to measure at CAS time. The I/O module or probe used to measure the RAS and CAS address activity would be synchronized to Pod Data Sync or Pod Address Sync. However, the *setoffset* command provides an offset from Pod Data Sync or Pod Address Sync that determines when the clocking for measurements actually occurs.

For some designs, more than one RAS cycle can occur during a read or write cycle. The bottom half of Figure 4-25 shows typical timing for such a situation. RAS goes low first for a refresh and then again later for the read. In this case, it is not sufficient to clock measurements on address lines with RAS alone. If you want to examine the row address signals on the address lines, you could use the Refresh signal to qualify clocking for the appropriate address information.

Measuring the RAS and CAS Lines

An easy check for RAS and CAS lines is to look for activity on the lines. With the probe or I/O module synchronized to the FREERUN clock, an asynchronous level history for RAS should always show high and low levels and never an invalid

level. An asynchronous level history for CAS will be the same as RAS if it is being accessed at the time. When the RAM is not being accessed, CAS may be similarly active or it may remain high, depending on the UUT.

Although the absence of the proper levels described above will indicate some types of faults, these simple checks cannot determine if the lines are definitely good. Subtle timing problems are common with some dynamic RAM designs.

To analyze the exact timing of RAS and CAS, use the 9100A/9105A to generate the appropriate sync signal and to display the UUT waveforms on an oscilloscope:

1. Use the SYNC key on the operator's keypad to select the Pod Address Synchronization mode:

   ```
   SYNC TO POD ADDR
   or SYNC I/O MOD <number> TO POD ADDR
   ```

2. Use the READ key on the operator's keypad to enter the following command:

   ```
   READ FAST FOREVER ADDR <ram address>
   ```

3. Synchronize an oscilloscope to the TRIGGER OUTPUT sync output on the rear panel of the 9100A/9105A.

4. Study the oscilloscope waveforms at the dynamic RAM chips.

Once the timing of RAS and CAS (as well as other dynamic RAM signals) is understood from the above procedure, two options are available. The first is to troubleshoot directly with a synchronized oscilloscope, and the second is to write a TL/1 program to automate the procedure.

Determining If Refresh Signals Are Working

Typical dynamic RAM must access every row address for cell refresh at least every 2 milliseconds. The ability of the 9100A/9105A to measure frequency min-max is the simplest tool for troubleshooting the circuitry that implements this refresh. No matter how the refresh circuitry is designed, the refresh signals (refresh address, RAS, and related timing signals) are on a regular schedule of one full cycle in less than 2 milliseconds. For a first-cut characterization of these signals, try measuring frequency min-max.

For a more precise characterization of the refresh signals, use the external synchronization capabilities (start, stop, clock) of the 9100A/9105A. Characterize all related signals during the start/stop interval of one refresh cycle, and then characterize the signals used for start/stop/clock with frequency min-max.

Dynamic RAM Timing Circuit Example 4.4.3.

A diagram of read/write timing for the Demo/Trainer UUT's RAM timing circuit is shown in Figure 4-26. The circuit schematic is shown in Figure 4-28.

Accessing

To select RAM, U65 and U66 multiplex 16 address lines into eight lines. The multiplexed address is then latched into the RAM chips by two externally applied clock pulses. The first, the negative-going edge of the row-address strobe (~RAS), latches the eight row-address bits. The second, the negative-going edge of the column-address strobe (~CAS), latches the eight column-address bits. Timing for RAS and CAS is determined by delay line U60. CAS is a delayed RAS signal; it goes low 55 nsec after RAS goes low.

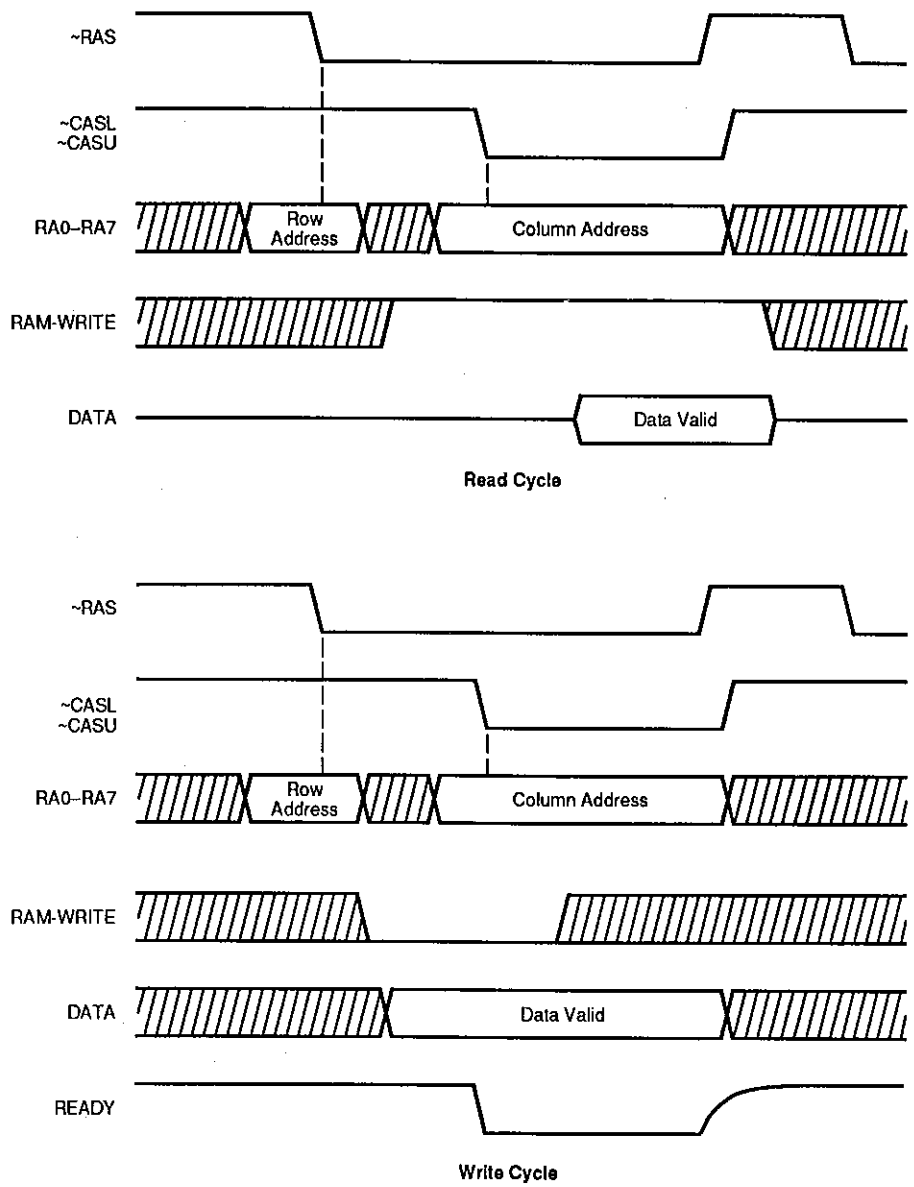


Figure 4-26: Dynamic RAM Read/Write Timing

The 80286 can access upper and lower bytes separately, or together as a word. RAM is organized as 128K bytes, addressed from 00000 to 1FFFE. Access is accomplished by gating ~CASL and ~CASU (U58D). IA00 (internal buffered address bit zero) selects DO-D7 and ~IBHE (Internal Buffered Bus High Enable) selects D8-D15. The low byte is accessed when IA00 is low. The high byte is accessed when IBHE- is low. The entire word is accessed when both IA00 and ~IBHE are low. The 80286 determines the type of access based on the instruction being executed.

Refreshing

RAM Refresh timing is illustrated in Figure 4-27.

To maintain data, each of the 128 RAS addresses must be refreshed (or read) every 2 msec. The Demo/Trainer UUT uses the RAS-only refresh method for this purpose. A RAS-only refresh cycle asserts only the RAS line to strobe in the refresh address.

A single Demo/Trainer UUT row refresh occurs every 15 µsec. A complete refresh entails 128 row refreshes, requiring about 1.9 msec.

The RFRQ (Refresh Request) signal both marks the need for a refresh cycle and increments the refresh address counter U67. U42 and U43 are used to divide PCLK (4 MHz) by 16 to produce RFRQ.

RAM refresh and RAM access are mutually exclusive. U61D insures that a refresh cannot occur if a RAM access is in progress. Conversely, if a refresh is in progress and the processor asks for a RAM access, U58B prevents Ready from being returned, causing the addition of a wait state. The processor is thus put on hold until the refresh is completed.

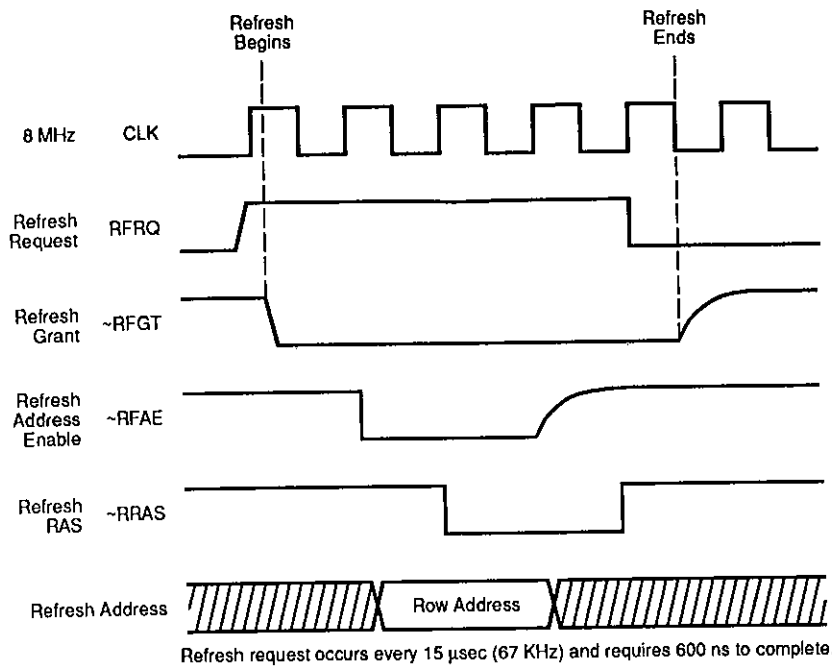


Figure 4-27: RAM Refresh Timing

RAM refresh is performed as follows:

1. If ~RAM is high (no RAM access in progress) and refresh is being requested, U61D outputs RFGT (Refresh Grant).

2. RFGT high enables the U44A/U44B state machine. This circuit times the output of Refresh Address Enable (RFAE) to U67. After the proper refresh address setup time, it also enables Refresh RAS (RRAS) to strobe in the refresh address.

3. After the refresh address is strobed in, RFGT goes low, allowing the processor access to the RAM.

Keystroke Functional Test 4.4.4.

1. Use a 16-pin clip module on side A of I/O module 1 to check CAS addresses and select line. Use the the EXEC and I/O MOD keys with the commands below for each of the following parts: U65, U66 and U26. The correct measurement for each pin is shown in the table below.

```
EXECUTE UUT DEMO PROGRAM CAS_STIM
SHOW I/O MOD 1 PIN <see table> CAPTURED ...
... RESPONSES
```

NOTE

The SHOW command requires a clip module pin number rather than a part pin number. This requires you to translate part pin numbers to clip module pin numbers (see Appendix B of the Technical User's Manual). For your convenience, this translation has been done for you in this example, and the results are shown in the "I/O MOD PIN" column of the response table in the next figure.

SIGNAL	*PART/PIN*	*I/O PIN*	*SIGNATURE*
RA0	U65-4	4	0140
RA1	-7	7	02AF
RA2	-9	13	0150
RA3	-12	16	03A9
RA4	U66-4	4	00D3
RA5	-7	7	022A
RA6	-9	13	0151
RA7	-12	16	0263
RAM-WRITE	U26-8	14	0352

2. Use a 16-pin clip module on side A of I/O module 1 to check RAS addresses. Use the the EXEC and I/O MOD keys with the commands below for each of the following parts: U65 and U66. The correct measurement for each pin is shown in the table below.

```
EXECUTE UUT DEMO PROGRAM RAS_STIM
SHOW I/O MOD 1 PIN <see table> CAPTURED ...
... RESPONSES
```

SIGNAL	*PART/PIN*	*I/O PIN*	*SIGNATURE*
RA0	U65-4	4	02BF
RA1	-7	7	0154
RA2	-9	13	022A
RA3	-12	16	01D1
RA4	U66-4	4	022A
RA5	-7	7	0150
RA6	-9	13	022B
RA7	-12	16	0114

3. The next step is measuring refresh signals that are active with no stimulus. Use a 16-pin clip module on side A of I/O module 1 to test refresh signals on RA0-RA7. Connect the external control lines as follows:

   Start to U67-9
   Stop to U67-9
   Clock to U63-8

Use the the SYNC and I/O MOD keys with the commands below to measure refresh signals. The correct measurement for each pin is shown in the table below.

```
SYNC I/O MOD 1 TO EXT ENABLE ALWAYS ...
... CLOCK ↓ START ↑ STOP ↑
ARM I/O MOD 1 FOR CAPTURE USING SYNC
SHOW I/O MOD 1 PIN <see table> CAPTURED ...
... RESPONSES
```

SIGNAL	*PART/PIN*	*I/O PIN*	*SIGNATURE*
RA0	U65-4	4	968C
RA1	-7	7	AFC1
RA2	-9	13	4A2C
RA3	-12	16	25AF
RA4	U66-4	4	ACDE
RA5	-7	7	122D
RA6	-9	13	EEA6
RA7	-12	16	68F8

4. Use a 14-pin clip module on side A of I/O module 1 to check the select logic. Use the the EXEC and I/O MOD keys with the commands below. The correct measurement for each pin is shown in the response table in Figure 4-28.

```
EXECUTE UUT DEMO PROGRAM RAMSELECT1
SHOW I/O MOD 1 PIN 14 CAPTURED RESPONSES
```

5. Use a 14-pin clip module on side A of I/O module 1 to check the select logic. Use the the EXEC and I/O MOD keys with the commands below. The correct measurement for each pin is shown in the response table in Figure 4-28.

```
EXECUTE UUT DEMO PROGRAM RAMSELECT2
SHOW I/O MOD 1 PIN 14 CAPTURED RESPONSES
SHOW I/O MOD 1 PIN 17 CAPTURED RESPONSES
```

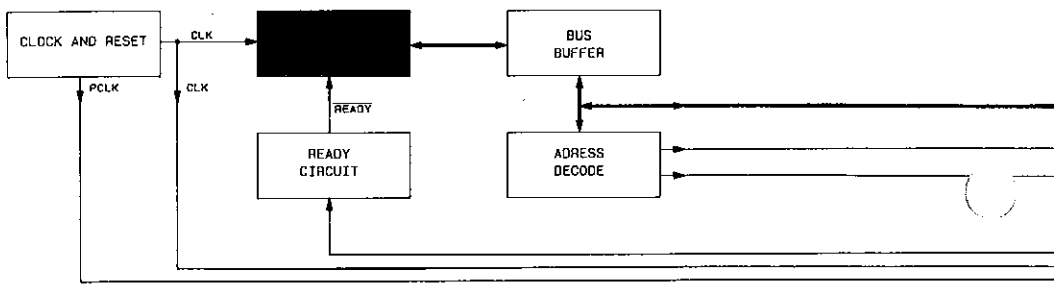
Keystroke Functional Test

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	I/O MOD U65 U26 U66 U63 U58

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
RA0	U65-4	4	SEE TEXT
RA1	-7	7	
RA2	-9	13	
RA3	-12	16	
RA4	U66-4	4	SEE TEXT
RA5	-7	7	
RA6	-9	13	
RA7	-12	16	
RAM-WRITE	U26-8	14	
RASS	U63-8	14	01B6
CASU	U58-8	14	B6FD
CASL	-11	17	B603



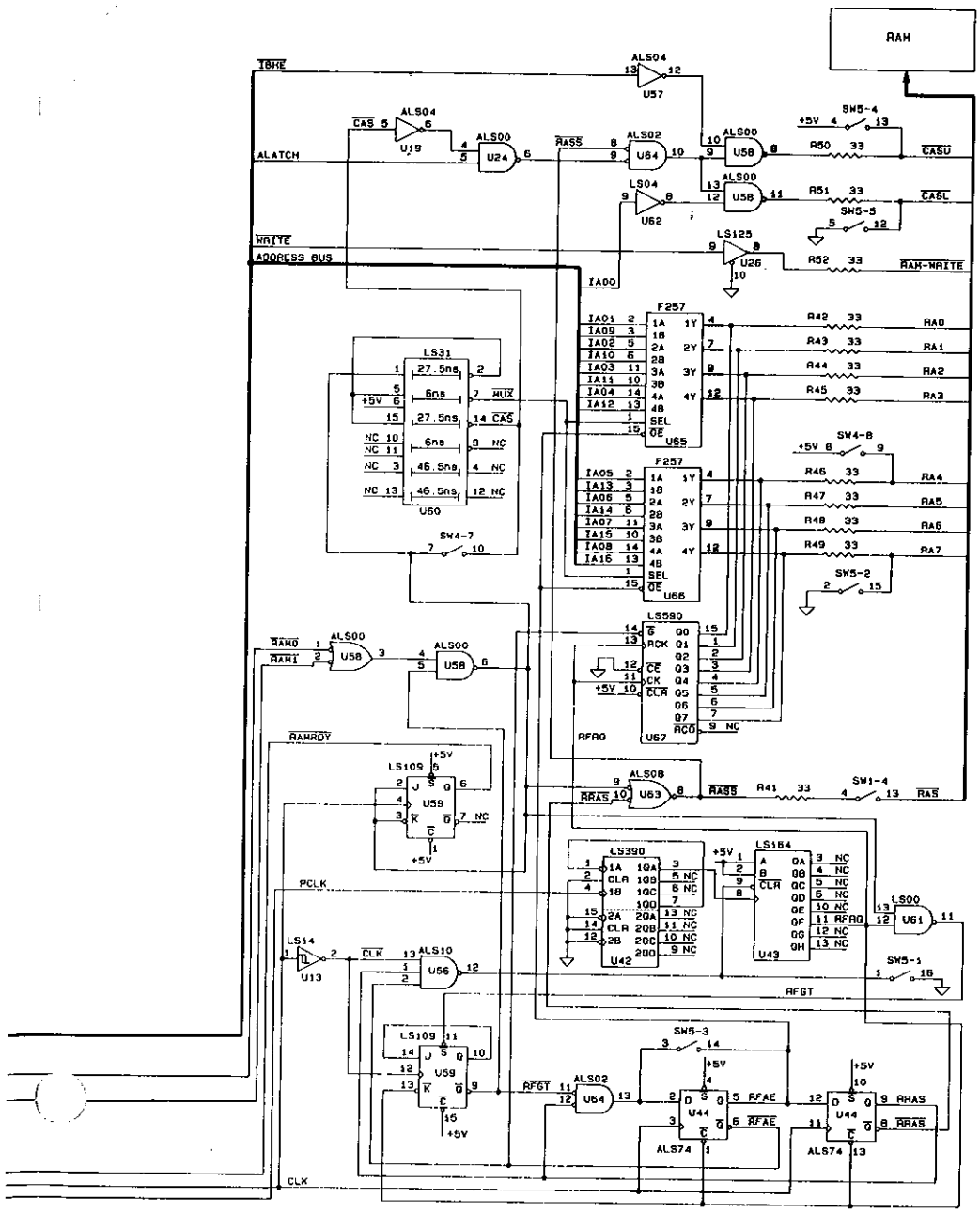


Figure 4-28: Dynamic RAM Timing Functional Test

Programmed Functional Test 4.4.5.

The *tst_refrsh* program is the programmed functional test for the Dynamic RAM Timing functional block. This program checks the outputs at U65, U58, U63 and U25 using the *gfi test* command. If the *gfi test* command fails, the *abort_test* program is executed and GFI troubleshooting begins. (See the Bus Buffer functional block for a discussion of the *abort_test* program).

```
program tst_refrsh

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the DYNAMIC RAM REFRESH functional block.             !
!                                                                         !
! This program tests the DYNAMIC RAM REFRESH functional block of the      !
! Demo/Trainer. The gfi test command and I/O module are used to perform   !
! the test.                                                               !
!                                                                         !
! TEST PROGRAMS CALLED:                                                   !
!    abort_test (ref-pin)                  If gfi has an accusation       !
!                                          display the accusation else    !
!                                          create a gfi hint for the      !
!                                          ref-pin and terminate the test!
!                                          program (GFI begins trouble-   !
!                                          shooting).                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   print "\nlTESTING RAM TIMING & REFRESH Circuit"

   podsetup 'enable ~ready' "on"

   if gfi test "U65-1" fails then abort_test("U65-1")
   if gfi test "U66-1" fails then abort_test("U66-1")

   print "RAM TIMING & REFRESH TEST PASSES"
end program
```

Stimulus Programs and Responses 4.4.6.

Figure 4-29 is the stimulus program planning diagram for the Dynamic RAM Timing functional block. The *ras_stim* and *cas_stim* stimulus programs both perform read and write accesses to various addresses in RAM. However, the *getoffset* and *setoffset* commands are used to adjust the timing when the data is measured, so that *cas_stim* measures data when CAS addresses are valid and *ras_stim* measure data when RAS addresses are valid.

The *ramselect1* and *ramselect2* programs provide stimulus for measurement of a number of logic outputs. The *refsh_addr*, *refsh_time*, and *refsh_u56* programs provide stimulus for measurement at various ICs that perform the RAM refresh function. The *frequency* program measures frequency at a number of nodes.

Stimulus Program Planning

PROGRAM: CAS_STIM

EXERCISES THE CAS ADDRESS

MEASUREMENT AT:

U65-4,7,9,12
U66-4,7,9,12
U26-8

[redacted]

[redacted]

PROGRAM: RAS_STIM

EXERCISES THE RAS ADDRESS

MEASUREMENT AT:

U65-4,7,9,12
U66-4,7,9,12

[redacted]

PROGRAM: RAMSELECT1

EXERCISES THE RAM SELECT LOGIC

MEASUREMENT AT:

U19-6	U58-3,6
U24-6	U59-6,10,9
U64-10	U63-8
U60-2,7,14	U61-11

PROGRAM: RAMSELECT2

EXERCISES THE RAM SELECT LOGIC

MEASUREMENT AT:

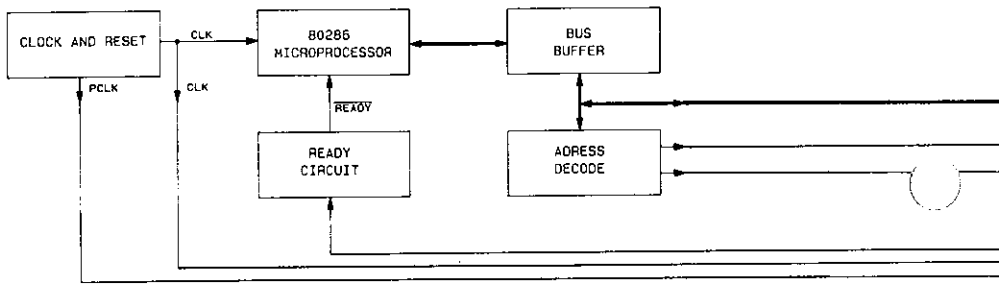
U57-12
U62-8
U58-8,11

PROGRAM: REFSH_U56

MEASURES REFRESH TIMING FOR U56

MEASUREMENT AT:

U56-12



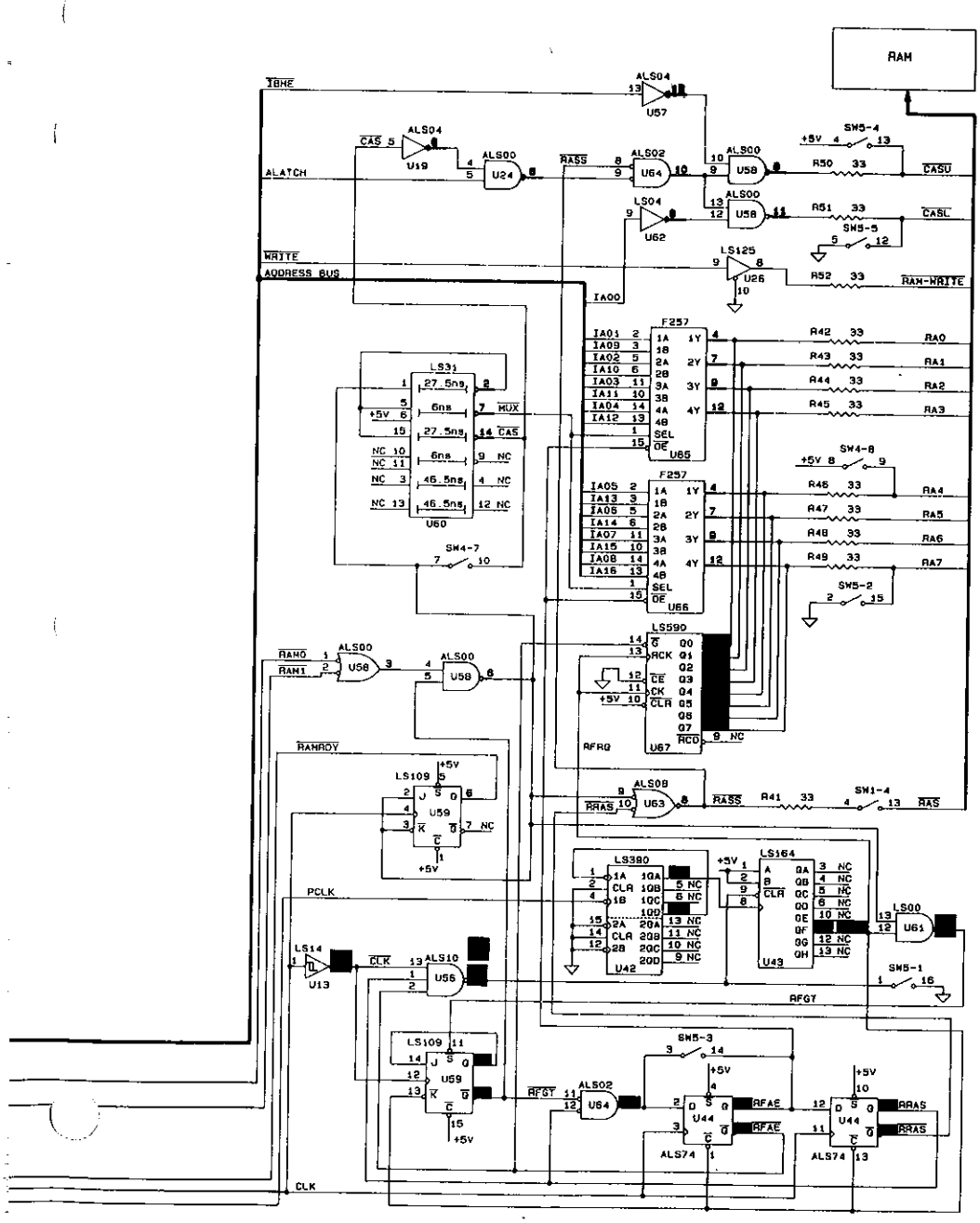


Figure 4-29: Dynamic RAM Timing Stimulus Program Planning

```
program cas_stim

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM characterizes CAS address lines.                         !
!                                                                           !
! Stimulus programs and response files are used by GFI to back-trace        !
! from a failing node.  The stimulus program must create repeatable UUT     !
! activity and the response file contains the known-good responses for      !
! the outputs in the UUT that are stimulated by the stimulus program.       !
!                                                                           !
! This stimulus program is one of the programs which creates activity       !
! in the RAM area of the UUT.  This stimulus program uses the setoffset     !
! and getoffset commands to adjust the timing to CAS address valid.         !
!                                                                           !
! TEST PROGRAMS CALLED:                                                     !
!                                                                           !
! GRAPHICS PROGRAMS CALLED:                                                 !
!    (none)                                                                 !
!                                                                           !
! Local Variables Modified:                                                 !
!    devname                              Measurement device                !
!    bias                                 Offset value to use               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare numeric bias = 999957

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program CAS_STIM"
```

(continued on the next page)

Figure 4-30: Stimulus Program *(cas_stim)*

```
! Set addressing mode and setup measurement device.

   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   setspace space (getspace space "memory", size "word")
   reset device devname
   sync device devname, mode "pod"
   sync device "/pod", mode "data"

! Store calibration offset, set new offset
! Display warning message if setting new offset fails

   cal_offset = getoffset device devname
   if (setoffset device devname, offset bias) = 0 then
      fault 'setoffset returned a bad status, fatal error'
   end if

! Present stimulus to UUT.

   arm device devname
      read addr $AB54          ! This addr gives complmentary CAS address
      read addr $1549A
      write addr $1234, data $4320
      read addr $55AA
      write addr $AB54, data $AAAA
      read addr $156A8
      write addr $AA54, data $55AA
      read addr $1AD50
      write addr $1FFFE, data $FFFE
      read addr $2AD4
   readout device devname

! Restore calibration offset

   setoffset device "/mod1", offset cal_offset
end cas_stim
```

Figure 4-30: Stimulus Program *(cas_stim) - continued*

```
STIMULUS PROGRAM NAME: CAS_STIM
DESCRIPTION:                                          SIZE:            199 BYTES

                  ------------------- Response Data --------------------
   Node      Learned                 Async Clk Counter                    Priority
Signal Src    With        SIG         LVL  LVL  Mode      Counter Range     Pin

U65-4       I/O MODULE  0140          1  0       TRANS
U65-7       I/O MODULE  02AF          1  0       TRANS
U65-9       I/O MODULE  0150          1  0       TRANS
U65-12      I/O MODULE  03A9          1  0       TRANS
U66-7       I/O MODULE  022A          1  0       TRANS
U66-9       I/O MODULE  0151          1  0       TRANS
U66-12      I/O MODULE  0263          1  0       TRANS
u66-4       I/O MODULE  00D3          1  0       TRANS
u26-8       I/O MODULE  0352          1  0       TRANS
```

Figure 4-31: Response File *(cas_stim)*

```
program ras_stim

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM characterizes RAS address lines.                        !
!                                                                          !
! Stimulus programs and response files are used by GFI to backtrace        !
! from a failing node.  The stimulus program must create repeatable UUT    !
! activity and the response file contains the known-good responses for    !
! the outputs in the UUT that are stimulated by the stimulus program.      !
!                                                                          !
! TEST PROGRAMS CALLED:                                                    !
!     (none)                                                               !
!                                                                          !
! GRAPHICS PROGRAMS CALLED:                                                !
!     (none)                                                               !
!                                                                          !
! Local Variables Modified:                                                !
!    devname                                   Measurement device          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    declare numeric bias = 999964

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the measurement device.

    if (gfi control) = "yes" then
        devname = gfi device
    else
        devname = "/mod1"
    end if
    print "Stimulus Program RAS_STIM"

! Set addressing mode and setup measurement device.

    setspace space (getspace space "memory", size "word")
    reset device devname
    sync device devname, mode "pod"
    sync device "/pod", mode "addr"
```

(continued on the next page)

Figure 4-32: Stimulus Program *(ras_stim)*

```
! Store calibration offset, set new offset
! Display warning message if setting new offset fails

    cal_offset = getoffset device devname
    if (setoffset device devname, offset bias) = 0 then
       fault 'setoffset returned a bad status, fatal error'
    end if

! Present stimulus to UUT.

   arm device devname
      read addr $AB54          ! This addr gives complementary CAS address
      read addr $1549A
      write addr $1234, data $4320
      read addr $55AA
      write addr $AB54, data $AAAA
      read addr $156A8
      write addr $AA54, data $55AA
      read addr $1AD50
      write addr $1FFFE, data $FFFE
      read addr $2AD4
   readout device devname

! Restore the calibrated offset value.

   setoffset device devname, offset cal_offset

end ras_stim
```

Figure 4-32: Stimulus Program *(ras_stim) - continued*

STIMULUS PROGRAM NAME: RAS_STIM
DESCRIPTION: SIZE: 182 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U65-4	I/O MODULE	0QBF	1	0	TRANS		
U65-7	I/O MODULE	0154	1	0	TRANS		
U65-9	I/O MODULE	022A	1	0	TRANS		
U65-12	I/O MODULE	01D1	1	0	TRANS		
U66-4	I/O MODULE	022A	1	0	TRANS		
U66-7	I/O MODULE	0150	1	0	TRANS		
U66-9	I/O MODULE	022B	1	0	TRANS		
U66-12	I/O MODULE	0114	1	0	TRANS		

Figure 4-33: Response File *(ras_stim)*

```
program ramselect1

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to wiggle RAM select circuitry.                         !
!                                                                          !
! Stimulus programs and response files are used by GFI to backtrace       !
! from a failing node.  The stimulus program must create repeatable UUT   !
! activity and the response file contains the known-good responses for    !
! the outputs in the UUT that are stimulated by the stimulus program.     !
!                                                                          !
! Ramselect1 is used to stimulate the RAM select circuitry after the      !
! decoders.  The stimulus is a combination of reads that will ensure      !
! the decoder and related circuitry is working properly.                  !
!                                                                          !
! TEST PROGRAMS CALLED:                                                    !
!    recover      ()                       The 80286 microprocessor has a!
!                                          bus controller that is totally!
!                                          separate from the pod.  In    !
!                                          some cases the pod can get out!
!                                          of sync with the bus control- !
!                                          ler.  The recover program     !
!                                          resynchronizes the pod and the!
!                                          bus controller.               !
!                                                                          !
! GRAPHICS PROGRAMS CALLED:                                                !
!    (none)                                                                !
!                                                                          !
! Global Variables Modified:                                               !
!    recover_times                         Reset to Zero                  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  FAULT HANDLERS:                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
handle pod_timeout_enabled_line
  recover()
end handle

handle pod_timeout_recovered
  recover()
end handle

! Let GFI determine the measurement device.

    if (gfi control) = "yes" then
        devname = gfi device
    else
        devname = "/mod1"
    end if
    print "Stimulus Program RAMSELECT1"
```

(continued on the next page)

Figure 4-34: Stimulus Program *(ramselect1)*

```
! Set addressing mode and setup measurement device.

    setspace space (getspace space "memory", size "word")
    reset device devname
    sync device devname, mode "pod"
    sync device "/pod", mode "data"

! Present stimulus to UUT.

    arm device devname
       read addr $1A5A4
       read addr $F0000
       read addr $F0000
       read addr $5A5A
       read addr $F0000
       read addr $F0000
       write addr $7BDE, data $1234
       read addr $F0000
       write addr $15A5A, data $9876
       read addr $F0000
  readout device devname

end program
```

Figure 4-34: Stimulus Program *(ramselect1) - continued*

STIMULUS PROGRAM NAME: RAMSELECT1
DESCRIPTION: SIZE: 267 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U58-3	I/O MODULE	024F	1	0	TRANS		
U58-6	I/O MODULE	01B6	1	0	TRANS		
U59-6	I/O MODULE	01B6	1	0	TRANS		
U61-11	I/O MODULE	03F9	1	0	TRANS		
U60-2	I/O MODULE	024F	1	0	TRANS		
U60-7	I/O MODULE	01B6	1	0	TRANS		
U60-14	I/O MODULE	01B6	1	0	TRANS		
U59-9	I/O MODULE	03F9	1	0	TRANS		
U63-8	I/O MODULE	01B6	1	0	TRANS		
U19-6	I/O MODULE	024F	1	0	TRANS		
U24-6	I/O MODULE	01B6	1	0	TRANS		
U64-10	I/O MODULE	024F	1	0	TRANS		
U59-10	I/O MODULE	0000	1	0	TRANS		

Figure 4-35: Response File *(ramselect1)*

```
program ramselect2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM characterizes RAM select logic.                          !
!                                                                           !
! Stimulus programs and response files are used by GFI to backtrace         !
! from a failing node.  The stimulus program must create repeatable UUT     !
! activity and the response file contains the known-good responses for      !
! the outputs in the UUT that are stimulated by the stimulus program.       !
!                                                                           !
! Ramselect2 is used to stimulate the RAM select circuitry after the        !
! decoders.  The stimulus is a combination of reads that will ensure        !
! the decoder and related circuitry is working properly.  Ramselect2        !
! differs for ramselect1 because setoffset is required to delay the         !
! data due to signal propogation though the number of parts in the          !
! ram decode circuitry.                                                     !
!                                                                           !
! TEST PROGRAMS CALLED:                                                     !
!     (none)                                                                !
!                                                                           !
! GRAPHICS PROGRAMS CALLED:                                                 !
!     (none)                                                                !
!                                                                           !
! Global Variables Modified:                                                !
!     (none)                                                                !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    declare numeric bias = 999957

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle
handle pod_timeout_no_clk
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the measurement device.

    if (gfi control) = "yes" then
        devname = gfi device
```

(continued on the next page)

Figure 4-36: Stimulus Program *(ramselect2)*

```
    else
        devname = "/mod1"
    end if
    print "Stimulus Program RAMSELECT2"

! Set addressing mode and setup measurement device.

    mem_word = getspace space "memory", size "word"
    mem_byte = getspace space "memory", size "byte"
    reset device devname
    sync device devname, mode "pod"
    sync device "/pod", mode "data"

! Store calibration offset, set new offset
! Display warning message if setting new offset fails

    cal_offset = getoffset device devname
    if (setoffset device devname, offset bias) = 0 then
       fault 'setoffset returned a bad status, fatal error'
    end if

! Present stimulus to UUT.

    arm device devname
       setspace (mem_word)
       read addr $1A5A4
       read addr $F0000
       read addr $F0000
       read addr $5A5A
       read addr $F0000
       read addr $F0000
       write addr $7BDE, data $1234
       read addr $F0000
       write addr $15A5A, data $9876
       read addr $F0000

       setspace (mem_byte)
       read addr 1
       read addr 2
       read addr 3
       write addr 4, data 0
       write addr 5, data $12
       read addr $1111
       read addr $11111
       read addr $AAAA
    readout device devname

! Restore original calibration offset

    setoffset device devname, offset cal_offset
end program
```

Figure 4-36: Stimulus Program *(ramselect2) - continued*

STIMULUS PROGRAM NAME: RAMSELECT2
DESCRIPTION: SIZE: 114 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
u58-8	I/O MODULE	B6FD	1	0	TRANS		
u58-11	I/O MODULE	B603	1	0	TRANS		
u62-8	I/O MODULE	F963	1	0	TRANS		
u57-12	I/O MODULE	F99D	1	0	TRANS		

Figure 4-37: Response File *(ramselect2)*

```
program refsh_addr

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM characterizes the refresh circuitry.                    !
!                                                                          !
! Stimulus programs and response files are used by GFI to backtrace       !
! from a failing node.  The stimulus program must create repeatable UUT   !
! activity and the response file contains the known-good responses for    !
! the outputs in the UUT that are stimulated by the stimulus program.     !
!                                                                          !
! TEST PROGRAMS CALLED:                                                    !
!    check_meas (device, start, stop, clock, enable)                       !
!                                          Checks to see if the measure-  !
!                                          ment is complete using the     !
!                                          TL/1 checkstatus command.  If  !
!                                          the measurement times out then!
!                                          redisplay connect locations.   !
!                                                                          !
! GRAPHICS PROGRAMS CALLED:                                                !
!    (none)                                                                !
!                                                                          !
! Local Variables Modified:                                                !
!    done                                  returned from check_meas()     !
!    devname                               Measurement device             !
!                                                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare numeric done = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program REFSH_ADDR"
```

(continued on the next page)

Figure 4-38: Stimulus Program *(refsh_addr)*

```
! Set addressing mode and setup measurement device.

   setspace space (getspace space "memory", size "word")
   reset device devname
   sync device devname, mode "ext"
   enable device devname, mode "always"
   edge device devname, start "+", stop "-", clock "-"

! Prompt user to connect external lines.

   connect device devname, start "U67-9", stop "U67-9", clock "U63-8", common "gnd"

! External lines determine measurement.
! check_meas times out and reprompts if external lines aren't connected

   loop until done = 1
      arm device devname
         done = check_meas(devname, "U67-9", "U67-9", "U63-8", "*")
      readout device devname
   end loop

end program
```

Figure 4-38: Stimulus Program *(refsh_addr) - continued*

```
STIMULUS PROGRAM NAME: REFSH_ADDR
DESCRIPTION:                                          SIZE:            182 BYTES

                  ------------------- Response Data --------------------
   Node            Learned                 Async Clk Counter                    Priority
Signal Src          With          SIG        LVL  LVL  Mode     Counter Range     Pin

u67-15          I/O MODULE    96EC       1  0        TRANS
u67-1           I/O MODULE    AFC1       1  0        TRANS
u67-2           I/O MODULE    4A2C       1  0        TRANS
u67-3           I/O MODULE    25AF       1  0        TRANS
u67-4           I/O MODULE    ACDE       1  0        TRANS
u67-5           I/O MODULE    122D       1  0        TRANS
u67-6           I/O MODULE    EEA6       1  0        TRANS
u67-7           I/O MODULE    68F8       1  0        TRANS
```

Figure 4-39: Response File *(refsh_addr)*

```
program refsh_time

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM characterizes the refresh timing.                           !
!                                                                              !
! Stimulus programs and response files are used by GFI to backtrace            !
! from a failing node.  The stimulus program must create repeatable UUT        !
! activity and the response file contains the known-good responses for         !
! the outputs in the UUT that are stimulated by the stimulus program.          !
!                                                                              !
! TEST PROGRAMS CALLED:                                                        !
!    check_meas (device, start, stop, clock, enable)                           !
!                                                 Checks to see if the measure- !
!                                                 ment is complete using the   !
!                                                 TL/1 checkstatus command.  If !
!                                                 the measurement times out then!
!                                                 redisplay connect locations.  !
!                                                                              !
! GRAPHICS PROGRAMS CALLED:                                                    !
!    (none)                                                                    !
!                                                                              !
! Local Variables Modified:                                                    !
!    done                                         returned from check_meas()   !
!    devname                                      Measurement device           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare numeric done = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program REFSH_TIME"
```

(continued on the next page)

Figure 4-40: Stimulus Program *(refsh_time)*

```
! Set addressing mode and setup measurement device.

   setspace space (getspace space "memory", size "word")
   reset device devname
   sync device devname, mode "ext"
   enable device devname, mode "always"
   edge device devname, start "+", stop "count", clock "-"
   stopcount device devname, count 48

! Prompt user to connect external lines.

      connect device devname, start "U67-13", clock "U13-1", common "gnd"

! External lines determine measurement.
! check_meas times out and reprompts if external lines aren't connected.

   loop until done = 1
      arm device devname
         done = check_meas(devname, "U67-13", "*", "U13-1", "*")
      readout device devname
   end loop

end program
```

Figure 4-40: Stimulus Program *(refsh_time) - continued*

```
STIMULUS PROGRAM NAME: REFSH_TIME
DESCRIPTION:                                          SIZE:          195 BYTES

                    ------------------- Response Data --------------------
   Node      Learned                    Async Clk Counter                    Priority
Signal Src    With          SIG          LVL  LVL  Mode     Counter Range     Pin

u59-9         I/O MODULE    159A         1  0      TRANS
u64-13        I/O MODULE    909A         1  0      TRANS
u44-5         I/O MODULE    87E6         1  0      TRANS
U44-6         PROBE         DE42         1  0      TRANS
u44-6         I/O MODULE    DE42         1  0      TRANS
u59-10        I/O MODULE    4C3E         1  0      TRANS
U44-9         PROBE         43F3         1  0      TRANS
u44-9         I/O MODULE    43F3         1  0      TRANS
u44-8         I/O MODULE    1A57         1  0      TRANS
u61-11        I/O MODULE                 1  0      TRANS
u43-11        I/O MODULE                 1  0      TRANS
```

Figure 4-41: Response File *(refsh_time)*

```
program refsh_u56

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM characterizes the refresh circuitry.                       !
!                                                                             !
! Stimulus programs and response files are used by GFI to backtrace          !
! from a failing node.  The stimulus program must create repeatable UUT      !
! activity and the response file contains the known-good responses for       !
! the outputs in the UUT that are stimulated by the stimulus program.        !
!                                                                             !
! TEST PROGRAMS CALLED:                                                       !
!    check_meas (device, start, stop, clock, enable)                         !
!                                        Checks to see if the measure-       !
!                                        ment is complete using the          !
!                                        TL/1 checkstatus command.  If       !
!                                        the measurement times out then!
!                                        redisplay connect locations.        !
!                                                                             !
! GRAPHICS PROGRAMS CALLED:                                                   !
!    (none)                                                                   !
!                                                                             !
! Local Variables Modified:                                                   !
!    done                                returned from check_meas()           !
!    devname                             Measurement device                   !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare numeric done = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program REFSH_U56"
```

(continued on the next page)

Figure 4-42: Stimulus Program *(refsh_u56)*

```
! Set addressing mode and setup measurement device.

   setspace space (getspace space "memory", size "word")
   reset device devname
   sync device devname, mode "ext"
   enable device devname, mode "always"
   edge device devname, start "+", stop "count", clock "+"
   stopcount device devname, count 48

! Prompt user to connect external lines.

   connect device "/mod1", start "U67-13", clock "U13-1", common "gnd"

! External lines determine measurement.
! check_meas times out and reprompts if external lines aren't connected.

   loop until done = 1
      arm device devname
         done = check_meas(devname, "U67-13", "*", "U13-1", "*")
      readout device devname
   end loop

end program
```

Figure 4-42: Stimulus Program *(refsh_u56) - continued*

```
STIMULUS PROGRAM NAME: REFSH_U56
DESCRIPTION:                                        SIZE:          63 BYTES

                    ------------------- Response Data --------------------
   Node      Learned                Async Clk Counter                    Priority
Signal Src    With        SIG       LVL  LVL  Mode      Counter Range     Pin

U56-12        PROBE                 1 0       TRANS      1
u56-12        I/O MODULE            1 0       TRANS      1
```

Figure 4-43: Response File *(refsh_u56)*

Summary of Complete Solution for Dynamic RAM Timing 4.4.7.

The entire set of programs and files needed to test and GFI troubleshoot the Dynamic RAM Timing functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Dynamic RAM Timing)

Programs (PROGRAM):

TST_REFRSH	Functional test	Section 4.4.5
CAS_STIM	Stimulus Program	Figure 4-30
RAS_STIM	Stimulus Program	Figure 4-32
RAMSELECT1	Stimulus Program	Figure 4-34
RAMSELECT2	Stimulus Program	Figure 4-36
REFSH_ADDR	Stimulus Program	Figure 4-38
FREQUENCY	Stimulus Program	Figure 4-117
REFSH_TIME	Stimulus Program	Figure 4-40
REFSH_U56	Stimulus Program	Figure 4-42

Stimulus Program Responses (RESPONSE):

CAS_STIM	Figure 4-31
RAS_STIM	Figure 4-33
RAMSELECT1	Figure 4-35
RAMSELECT2	Figure 4-37
REFSH_ADDR	Figure 4-39
FREQUENCY	Figure 4-118
REFSH_TIME	Figure 4-41
REFSH_U56	Figure 4-43

Node List (NODE):

NODELIST	Appendix A

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix B

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

(This page is intentionally blank.)

PARALLEL INPUT/OUTPUT FUNCTIONAL BLOCK 4.5.

Introduction to Parallel I/O 4.5.1.

Parallel I/O implementations range in complexity from simple latches to LSI components. This section covers two basic types of parallel I/O circuits, simple discrete I/O circuits, and common LSI components like Programmable Interface Adapters (PIA) and Programmable Interval Timers (PIT).

Parallel I/O is one of a microcomputer's interfaces to the real world. The microcomputers in products like cash registers, copiers, telephone switching equipment, electronic instruments, and personal computers often monitor and control optical or electromechanical components like LEDs, displays, keyboards, optical switches, printers, disk or tape drives. Often, the interface to these components from the microprocessor's perspective is a set of registers to which it can read and write data.

Output lines may be connected to recording or display devices, which can be damaged if random data is written indiscriminately to them. Signals controlled by output ports can produce voltages or actuate devices that can pose a threat to human safety. Care should be taken in designing stimulus programs when the possibility of injury to people or damage to equipment can result.

Considerations for Testing and Troubleshooting 4.5.2.

Programmable LSI Components

Programmable LSI components usually contain internal registers which characterize the component to a particular circuit application. Among the ways in which these components can be programmed are:

- Set internal operating modes.
- Configure I/O ports as inputs or outputs.
- Set edge polarity on edge-sensitive inputs.
- Enable or disable interrupts.
- Establish data exchange protocol.

When testing LSI components, it is necessary to initialize them first. Initialization usually consists of a series of reads from and writes to internal registers. It is useful to create a separate 9100A initialization program which can be called from various stimulus programs, or from the operator's keyboard.

If a component, such as a PIA, does not work properly after initialization, check the inputs that affect its operation, such as chip-select lines, read and write lines, register-select lines, and clocks. Signals that reset, gate, or set outputs to high impedance might also be suspect. If these inputs all appear good, the bus cycles accessing the component may not have the proper number of wait states.

To verify operation of the component, stimulus commands such as *rampdata, read,* and *write* can be used in combination with I/O-module measurements. For troubleshooting both inputs and outputs on devices such as LEDs and keyboards, it is often necessary to prompt the operator to interact with the UUT. Simple commands prompting operator action can be included in stimulus programs and displayed on the operator's display.

Outputs can be tested with *write*, *toggledata*, or *rampdata* commands. Responses can be read as signatures or as asynchronous or clocked level history. Signatures are useful for identifying outputs that are tied to each other. If there is not an appropriate clock available, transition counts or level history can be used.

Inputs can be verified by reading the component. To exercise all states of the input lines, some type of stimulus must be applied. If the circuit allows, the inputs can be overdriven to each logic

state with the I/O module. For electromechanical devices such as keys and switches, interaction with the person performing a test may be required. Switch testing can be automated by using solenoids to actuate the switches.

Discrete I/O

Components used for discrete I/O include buffers, latches, addressable latches, and flip-flops. Such components usually have simpler interfaces to the microprocessor than programmable LSI components and they are handled in a similar manner, but their initialization procedures are different, if required at all.

If data does not appear to be reaching I/O latches, or is not read from I/O buffers, it may be necessary to check the address decoding logic to verify that the proper control signals are present. Here are some common problems associated with discrete I/O:

- *Outputs* may be loaded by external devices. Such outputs may work properly when disconnected. The loading problem may be associated with the external device, or with its connector.
- *Inputs* may be damaged by static electricity when they are disconnected from the signal sources and left unprotected.
- *Clocked inputs* on components like latches or flip-flops may be faulty.
- *Reset inputs* may either be stuck, forcing outputs to some state, or open, preventing circuits from being initialized.
- *Pullup or pulldown resistors* that establish static logic levels may be open, creating indeterminate inputs.

Parallel I/O Example 4.5.3.

The Programmable Interface Adapter on the Demo/Trainer UUT (U31) is shown in Figure 4-44. It can be programmed for operation with three ports, each with eight data lines. Each port is addressed for read or write by address lines IA01 and IA02. Ports A (lines PAO-7) and B (lines PB0-7) are used for outputs to the two on-board seven-segment LEDs. Port A corresponds to the upper LED, port B corresponds to the lower LED, and port C (lines PC0-7) is used for inputs from the four push-button switches.

Keystroke Functional Test 4.5.4.

Part A:

1. Initialize the Parallel I/O functional block using the WRITE key with the following commands:

```
WRITE DATA 89 TO ADDR 4006
... (ADDR OPTION: I/O BYTE)
WRITE DATA FF TO ADDR 4000
... (ADDR OPTION: I/O BYTE)
WRITE DATA FF TO ADDR 4002
... (ADDR OPTION: I/O BYTE)
```

2. Use the WRITE key to write values to the PIA chip. Read the resulting numbers on LED A. The values to be written and the results to be displayed are shown in the Response table in Figure 4-44.

```
WRITE DATA <see response table> TO ADDR 4000
... (ADDR OPTION: I/O BYTE)
```

3. Now use the WRITE key to write values to the PIA chip to display numbers on LED B. The values to be written and the results to be displayed are shown in the Response table in Figure 4-44.

```
WRITE DATA <see response table> TO ADDR 4002
... (ADDR OPTION: I/O BYTE)
```

Part B:

1. Use the READ key to read values resulting from pressing the UUT keys 1 through 4. The response table in Figure 4-45 shows the values that should be read for each key pressed.

```
READ ADDR 4004 = <see response table>
... (ADDR OPTION: I/O BYTE)
```

Keystroke Functional Test (Part A)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	VISUAL INSPECTION LEDA LEDB

STIMULUS AND RESPONSE TABLE FOR LEDA

[redacted]	LEDA SHOWS
	0 .1 2 .3 4 .5

STIMULUS AND RESPONSE TABLE FOR LEDB

[redacted]	LEDB SHOWS
	0 .1 .2 .3 4 .5

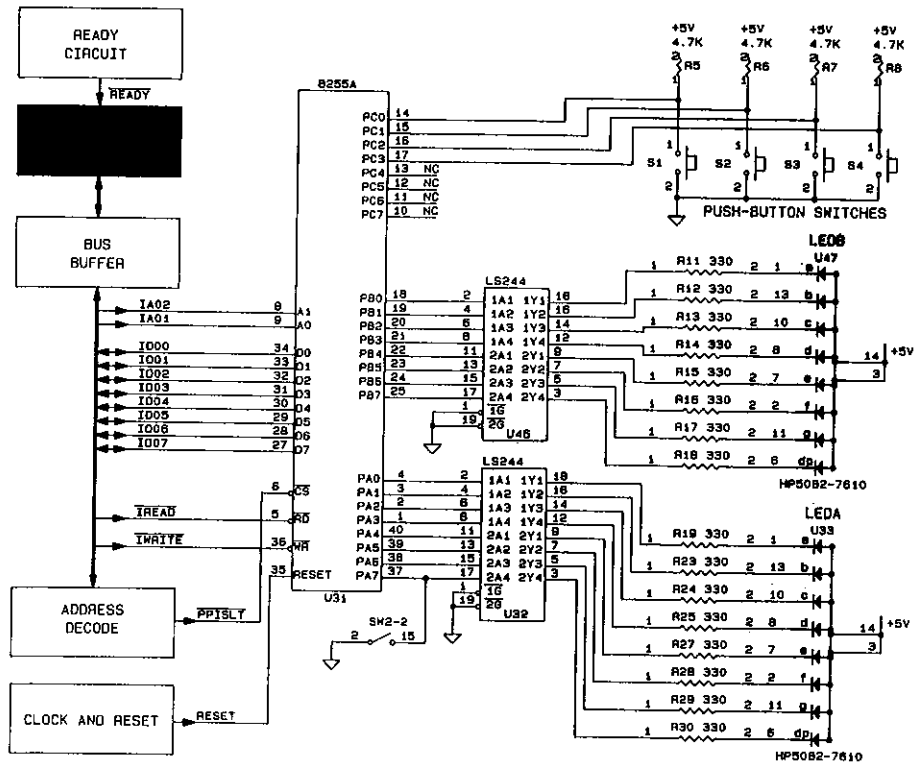


Figure 4-44: Parallel I/O Functional Test (Part A)

Keystroke Functional Test (Part B)

CONNECTION TABLE

[redacted]	[redacted]
[redacted] 51 52 53 54	POD TEST ACCESS SOCKET

STIMULUS AND RESPONSE TABLE FOR LEDA

[redacted]	DATA READ AT ADDRESS 4004
	F E D B 7 0

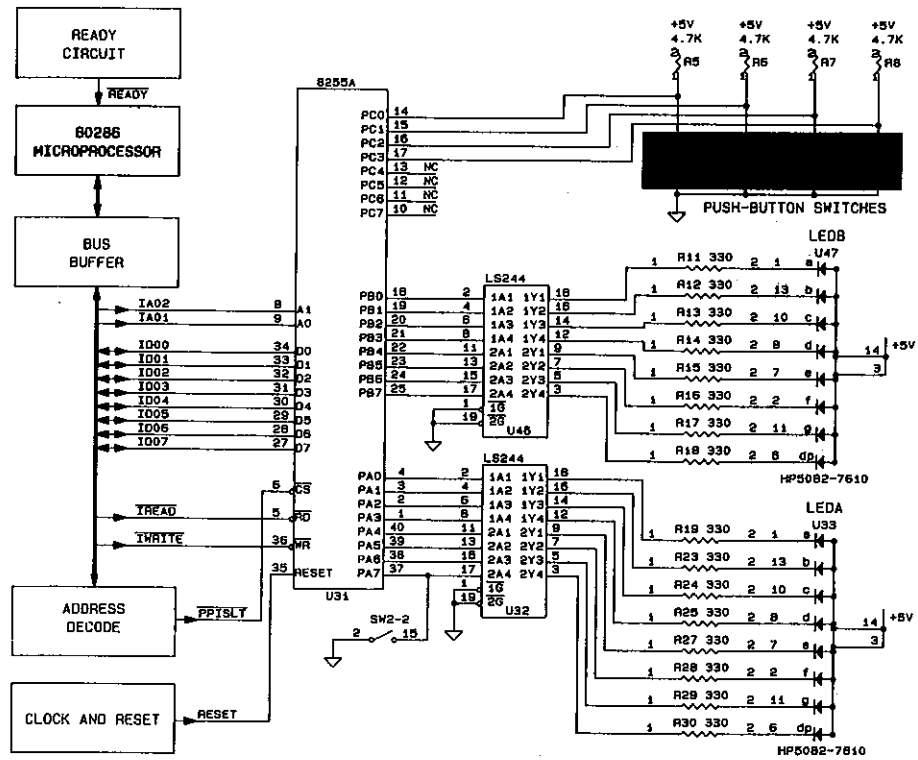


Figure 4-45: Parallel I/O Functional Test (Part B)

Programmed Functional Test 4.5.5.

The *test_pia* program is the programmed functional test for the Parallel I/O functional block. The program asks the test operator to check the visual properties of the LEDs that are driven by the PIA chip and also to check the mechanical operation of the pushbutton switches.

The program displays a message to the operator to watch LED A while the program displays numbers 1 through 9 on it. The operator is prompted to acknowledge proper operation or failing operation. If the LED fails, the *gfi test* command is used to test the LED drivers. If the LED drivers fail, GFI takes control and backtraces to the source of the failure. The same operation is then repeated for LED B.

Next, the operator is prompted to press key 1. The program polls the PIA chip and determines when the operator has pushed the key 1 button (if the key and the PIA are working properly). If the PIA cannot sense that the operator has pressed the key, the operator is instructed to press a 9100A/9105A key to indicate a failure. When the operator indicates a failing key, the *gfi test* command is used to verify correct signal levels at the key output. If a failure exists, GFI takes control and backtraces to the source of the failure. The same operation is repeated for keys 2, 3 and 4.

```
program test_pia

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the PARALLEL I/O functional block.                      !
!                                                                            !
! This program tests the PARALLEL I/O functional block of the                !
! Demo/Trainer.  The two LEDs and the four pushbutton switches are           !
! tested.  The test operator is prompted to visually inspect the LEDs        !
! as the LEDs count a series of numbers.                                     !
!                                                                            !
! TEST FUNCTIONS CALLED:                                                     !
!    keys        (key_number)              Test Demo/Trainer pushbutton      !
!                                          key key_number.  Prompt test      !
!                                          operator to push the key.         !
!                                                                            !
!    leds        (led_addr, led_name)      Test Demo/Trainer LED led_name!
!                                          which is driven by the PIA and!
!                                          has the address led_addr.         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Functions                                                               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

function keys(keynum)
   declare numeric keynum                          ! Number of key to test.
   declare string norm = "\1B[0m"                  ! Normal video escape string
   declare string rev  = "\1B[0;7m"                ! Reverse video escape string
   declare string entry
   declare string fail = ""
   declare global numeric tlb
   declare global numeric tli

   mask = setbit(keynum - 1)

   loop until fail = chr($D)                            ! loop until YES key
      print on tlb ,"\nlPress ", rev," UUT KEY ", keynum," ",norm," pushbutton"
      print on tlb ,"Press any 9100 key if test is stuck"
      loop until (poll channel tli, event "input") = 1
         if ((read addr $4004) and mask) = 0 then return
      end loop
      loop until (poll channel tli, event "input") = 0    ! Flush input buffer
         input on tli ,entry
      end loop
      print on tlb ,"\nlPress ",rev," YES ",norm," to fail KEY ",keynum," test,"
      print on tlb ,"Press "+rev+" NO  "+norm+" to continue key test,"
      input on tli ,fail
   end loop
   print on tlb ,"\nl\nl"
   fault                                    ! Fail Key test (set termination
end function                                ! status of function to fail.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

function leds(led_addr, led_name)
   declare numeric led_addr
   declare string led_name
   declare string key
   declare string norm = "\1B[0m"
   declare string bold = "\1B[1m"
   declare string rev = "\1B[7m"
   declare string clear_screen = "\1B[2J"
   declare string no_auto_linefeed = "\1B[20h"
   declare global numeric tli
   declare numeric array [0:10] numbers

   numbers [0] = $C0    \    numbers [5] = $92
   numbers [1] = $F9    \    numbers [6] = $82
   numbers [2] = $A4    \    numbers [7] = $F8
   numbers [3] = $B0    \    numbers [8] = $80
   numbers [4] = $99    \    numbers [9] = $98
   NO = chr($7F)        \    YES = chr( $D)

   print norm, clear_screen, "Watch LED ", led_name, " count"
   print "Press ", rev, " ENTER ", norm, " key to start LED counting."
   input  key
   print clear_screen

   for i = 0 to 9
      write addr led_addr, data numbers [i]
      wait time 500
   next
```

```
   write addr led_addr, data $7F
   print  clear_screen, "\1B[20l"
   print  "\1B[1;1fDid LED ", led_name, " display ALL segments off, then"
   print  "\1B[2;1fdigits 0 to 9, then only the Decimal Point ?"
   print  "\1B[3;fpress: "+rev+" YES "+norm+" or "+rev+" NO  "+norm
   loop until key = YES or key = NO
      input on tli ,key
      if key = NO then fault
   end loop
   write addr led_addr, data $FF \  print  no_auto_linefeed,clear_screen

end function

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! PARALLEL I/O Test.                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   tlb = open device "/term1", as "update", mode "buffered"
   tli = open device "/term1", as "input", mode "unbuffered"
   execute pia_init()

   if leds($4000, "A") fails then fault 'PIA LED A failed' \ return
   if leds($4002, "B") fails then fault 'PIA LED B failed' \ return

   if keys(1) fails then fault 'PIA KEY 1 failed' \ return
   if keys(2) fails then fault 'PIA KEY 2 failed' \ return
   if keys(3) fails then fault 'PIA KEY 3 failed' \ return
   if keys(4) fails then fault 'PIA KEY 4 failed' \ return

end program
```

Stimulus Programs and Responses 4.5.6.

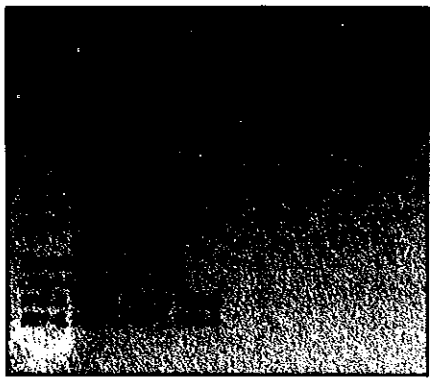
Figure 4-46 is the stimulus program planning diagram for the Parallel I/O functional block. The Parallel I/O stimulus programs only measure the electrical parameters of the Parallel I/O circuit; the visual properties of the LEDs are not measured.

The *ram_data* stimulus program outputs data from the PIA onto the data bus. The *pia_leds* stimulus program exercises outputs going to the LEDs. The *key_1, key_2, key_3,* and *key_4* stimulus programs monitor the operation of the four numbered pushbutton switches.

All the stimulus programs execute the *pia_init* program before any measurements are made on the PIA circuitry.

(This page is intentionally blank.)

Stimulus Program Planning



INITIALIZATION PROGRAM: PIA_INIT
INITIALIZES THE PIA PORT
MEASUREMENT AT:
(NONE)

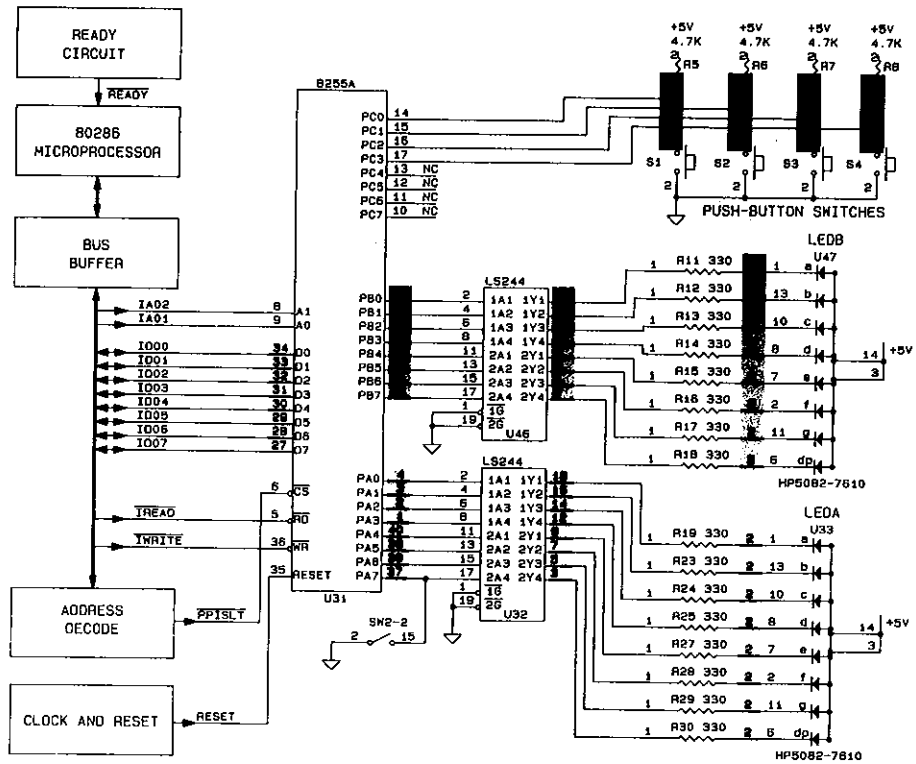


Figure 4-46: Parallel I/O Stimulus Program Planning

```
program key_1

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM checks KEY 1 of PIA circuit.                         !
!                                                                       !
! Stimulus programs and response files are used by GFI to backtrace     !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    pia_init   ()                                                      !
!                                                                       !
! GRAPHICS PROGRAMS CALLED:                                             !
!    (none)                                                             !
!                                                                       !
! Local Constants Modified:                                             !
!    CARRAGE_RETURN              Matches a carrage return input.        !
!                                                                       !
! Local Variables Modified:                                             !
!    devname                     Measurement device                     !
!    input_str                   Input from keypad                      !
!    state                       Level returned from measurement        !
!    finished                    State of loop looking for condition    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare global numeric finished = 0
   declare string CARRAGE_RETURN = ""
   declare string input_str
   declare numeric state = 0
   declare numeric high = 4
   finished = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Main part of STIMULUS PROGRAM                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the testing device.

   if (gfi control) = "yes" then
      devname = gfi device
      if (gfi ref) = "U31" then pinnum = 14
   else
      devname = "/probe"
   end if
   print "Stimulus Program KEY_1"
```

(continued on the next page)

Figure 4-47: Stimulus Program *(key_1)*

```
! Setup measurement device and prompt operator.

   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   reset device devname
   execute pia_init()
   setspace space (getspace space "i/o", size "byte")
   sync device devname, mode "int"
   tlup = open device "/term1", as "update"

! Wait for a high.  Leave program if <ENTER> key is pressed.

   loop until state = high
      arm device devname \ readout device devname
      if devname = "/probe" then
         state = level device devname, type "async"
      else
         state = level device devname, pin pinnum, type "async"
      end if
      if (poll channel tlup, event "input") = 1 then
         input on tlup ,input_str
         if input_str = CARRAGE_RETURN then return
      end if
   end loop

! Start response capture.  End when POD detects reset.
   arm device devname
      strobeclock device devname
      print on tlup ,"WHILE MEASURING, Press \1B[7mDemo UUT KEY 1\1B[0m"
      print on tlup ,"Press 9100 ENTER key if test is stuck."
      loop until finished = 1
         if ((read addr $4004) and 1) = 0 then
            wait time 2                          ! De-bounce.
            strobeclock device devname
            finished = 1
         else if (poll channel tlup, event "input") = 1 then
            input on tlup ,input_str
            if input_str = CARRAGE_RETURN then finished = 1
         end if
      end loop
   readout device devname

   print "\n1\n1"
end program
```

Figure 4-47: Stimulus Program *(key_1) - continued*

```
STIMULUS PROGRAM: KEY_1
DESCRIPTION:                                        SIZE:              78 BYTES

                  ------------------ Response Data --------------------
   Node            Learned                Async Clk Counter                    Priority
Signal Src         With          SIG       LVL  LVL  Mode     Counter Range     Pin

R5-1               PROBE         0002           1  0 TRANS
R5-1               I/O MODULE    0002           1  0 TRANS
```

Figure 4-48: Response File *(key_1)*

```
program key_2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM checks KEY 2 of PIA circuit.                            !
!                                                                          !
! Stimulus programs and response files are used by GFI to backtrace        !
! from a failing node.  The stimulus program must create repeatable UUT    !
! activity and the response file contains the known-good responses for     !
! the outputs in the UUT that are stimulated by the stimulus program.      !
!                                                                          !
! TEST PROGRAMS CALLED:                                                    !
!    pia_init  ()                                                          !
!                                                                          !
! GRAPHICS PROGRAMS CALLED:                                                !
!    (none)                                                                !
!                                                                          !
! Local Constants Modified:                                                !
!    CARRAGE_RETURN                    Matches a carrage return input.     !
!                                                                          !
! Local Variables Modified:                                                !
!    devname                           Measurement device                  !
!    input_str                         Input from keypad                   !
!    state                             Level returned from measurement     !
!    finished                          State of loop looking for condition !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  declare global numeric finished = 0
  declare string carrage_return = ""
  declare string str
  declare numeric state = 0
  declare numeric high = 4
  finished = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Main part of STIMULUS PROGRAM                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the testing device.

  if (gfi control) = "yes" then
     devname = gfi device
     if (gfi ref) = "U31" then pinnum = 15
  else
     devname = "/probe"
  end if
  print "Stimulus Program KEY_2"
```

(continued on the next page)

Figure 4-49: Stimulus Program *(key_2)*

```
! Setup measurement device and prompt operator.

   reset device devname
   execute pia_init()
   setspace space (getspace space "i/o", size "byte")
   sync device devname, mode "int"
   tlup = open device "/term1", as "update"

! Wait for a high.  Leave program if <ENTER> key is pressed.

   loop until state = high
      arm device devname \ readout device devname
      if devname = "/probe" then
         state = level device devname, type "async"
      else
         state = level device devname, pin pinnum, type "async"
      end if
      if (poll channel tlup, event "input") = 1 then
         input on tlup ,str
         if str = carrage_return then return
      end if
   end loop

! Start response capture.  End when PIA detects line low.

   arm device devname
      strobeclock device devname
      print on tlup ,"WHILE MEASURING, Press \1B[7mDemo UUT KEY 2\1B[0m"
      print on tlup ,"Press 9100 ENTER key if test is stuck."
      loop until finished = 1
         if ((read addr $4004) and 2) = 0 then
            wait time 2                          ! De-bounce.
            strobeclock device devname
            finished = 1
         else if (poll channel tlup, event "input") = 1 then
            input on tlup ,str
            if str = carrage_return then finished = 1
         end if
      end loop
   readout device devname

   print "\nl\nl"
end program
```

Figure 4-49: Stimulus Program *(key_2) - continued*

```
STIMULUS PROGRAM: KEY_2
DESCRIPTION:                                          SIZE:          78 BYTES

                  ------------------- Response Data --------------------
   Node            Learned                Async Clk Counter                    Priority
Signal Src          With          SIG       LVL  LVL  Mode     Counter Range     Pin

R6-1               PROBE         0002           1   0  TRANS
R6-1               I/O MODULE    0002           1   0  TRANS
```

Figure 4-50: Response File *(key_2)*

```
program key_3

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM checks KEY 3 of PIA circuit.                         !
!                                                                       !
! Stimulus programs and response files are used by GFI to backtrace     !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    pia_init   ()                                                      !
!                                                                       !
! GRAPHICS PROGRAMS CALLED:                                             !
!    (none)                                                             !
!                                                                       !
! Local Constants Modified:                                             !
!    CARRAGE_RETURN               Matches a carrage return input.       !
!                                                                       !
! Local Variables Modified:                                             !
!    devname                      Measurement device                    !
!    input_str                    Input from keypad                     !
!    state                        Level returned from measurement       !
!    finished                     State of loop looking for condition   !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare global numeric finished = 0
   declare string carrage_return = ""
   declare string str
   declare numeric state = 0
   declare numeric high = 4
   finished = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Main part of STIMULUS PROGRAM                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the testing device.

   if (gfi control) = "yes" then
      devname = gfi device
      if (gfi ref) = "U31" then pinnum = 16
   else
      devname = "/probe"
   end if
   print "Stimulus Program KEY_3"
```

(continued on the next page)

Figure 4-51: Stimulus Program *(key_3)*

```
! Setup measurement device and prompt operator.

   reset device devname
   execute pia_init()
   setspace space (getspace space "i/o", size "byte")
   sync device devname, mode "int"
   tlup = open device "/term1", as "update"

! Wait for a high.  Leave program if <ENTER> key is pressed.

   loop until state = high
      arm device devname \ readout device devname
      if devname = "/probe" then
         state = level device devname, type "async"
      else
         state = level device devname, pin pinnum, type "async"
      end if
      if (poll channel tlup, event "input") = 1 then
         input on tlup ,str
         if str = carrage_return then return
      end if
   end loop

! Start response capture.  End when POD detects reset.
   arm device devname
      strobeclock device devname
      print on tlup ,"WHILE MEASURING, Press \1B[7mDemo UUT KEY 3\1B[0m"
      print on tlup ,"Press 9100 ENTER key if test is stuck."
      loop until finished = 1
         if ((read addr $4004) and 4) = 0 then
            wait time 2                          ! De-bounce.
            strobeclock device devname
            finished = 1
         else if (poll channel tlup, event "input") = 1 then
            input on tlup ,str
            if str = carrage_return then finished = 1
         end if
      end loop
   readout device devname

   print "\n1\n1"
end program
```

Figure 4-51: Stimulus Program *(key_3) - continued*

```
STIMULUS PROGRAM: KEY_3
DESCRIPTION:                                          SIZE:              78 BYTES

                  ------------------- Response Data --------------------
   Node            Learned                Async Clk Counter                      Priority
Signal Src          With          SIG       LVL  LVL  Mode      Counter Range      Pin

R7-1               PROBE         0002          1  0  TRANS
R7-1               I/O MODULE    0002          1  0  TRANS
```

Figure 4-52: Response File *(key_3)*

```
program key_4

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM checks KEY 4 of PIA circuit.                                 !
!                                                                               !
! Stimulus programs and response files are used by GFI to backtrace             !
! from a failing node.  The stimulus program must create repeatable UUT        !
! activity and the response file contains the known-good responses for         !
! the outputs in the UUT that are stimulated by the stimulus program.           !
!                                                                               !
! TEST PROGRAMS CALLED:                                                         !
!    pia_init  ()                                                               !
!                                                                               !
! GRAPHICS PROGRAMS CALLED:                                                     !
!    (none)                                                                     !
!                                                                               !
! Local Constants Modified:                                                     !
!    CARRAGE_RETURN               Matches a carrage return input.               !
!                                                                               !
! Local Variables Modified:                                                     !
!    devname                      Measurement device                            !
!    input_str                    Input from keypad                             !
!    state                        Level returned from measurement               !
!    finished                     State of loop looking for condition           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare global numeric finished = 0
   declare string carrage_return = ""
   declare string str
   declare numeric state = 0
   declare numeric high = 4
   finished = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the testing device.

   if (gfi control) = "yes" then
      devname = gfi device
      if (gfi ref) = "U31" then pinnum = 17
   else
      devname = "/probe"
   end if
   print "Stimulus Program KEY_4"
```

(continued on the next page)

Figure 4-53: Stimulus Program *(key_4)*

```
! Setup measurement device and prompt operator.

   reset device devname
   execute pia_init()
   setspace space (getspace space "i/o", size "byte")
   sync device devname, mode "int"
   tlup = open device "/term1", as "update"

! Wait for a high.  Leave program if <ENTER> key is pressed.

   loop until state = high
      arm device devname \ readout device devname
      if devname = "/probe" then
         state = level device devname, type "async"
      else
         state = level device devname, pin pinnum, type "async"
      end if
      if (poll channel tlup, event "input") = 1 then
         input on tlup ,str
         if str = carrage_return then return
      end if
   end loop

! Start response capture.  End when BOD detects reset.
   arm device devname
      strobeclock device devname
      print on tlup ,"WHILE MEASURING, Press \1B[7mDemo UUT KEY 4\1B[0m"
      print on tlup ,"Press 9100 ENTER key if test is stuck."
      loop until finished = 1
         if ((read addr $4004) and 8) = 0 then
            wait time 2                          ! De-bounce.
            strobeclock device devname
            finished = 1
         else if (poll channel tlup, event "input") = 1 then
            input on tlup ,str
            if str = carrage_return then finished = 1
         end if
      end loop
   readout device devname

   print "\nl\nl"
end program
```

Figure 4-53: Stimulus Program *(key_4) - continued*

```
STIMULUS PROGRAM NAME: KEY_4
DESCRIPTION:                                          SIZE:          78 BYTES

                  ------------------- Response Data --------------------
   Node           Learned                 Async Clk Counter                    Priority
Signal Src        With          SIG       LVL  LVL  Mode     Counter Range     Pin

R8-1              PROBE         0002            1  0 TRANS
R8-1              I/O MODULE    0002            1  0 TRANS
```

Figure 4-54: Response File *(key_4)*

```
program pia_data

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM outputs data from the PIA onto the bus.                    !
!                                                                             !
! Stimulus programs and response files are used by GFI to backtrace          !
! from a failing node.  The stimulus program must create repeatable UUT      !
! activity and the response file contains the known-good responses for       !
! the outputs in the UUT that are stimulated by the stimulus program.        !
!                                                                             !
! This stimulus program is one of the programs which creates activity        !
! in the kernel area of the UUT.  These programs create activity with        !
! or without the ready circuit working properly.  Because of this, all       !
! the stimulus programs in the kernel area must disable the READY input      !
! to the pod, then perform the stimulus, then re-enable the READY input      !
! to the pod.  The 80286 microprocessor has a separate bus controller;       !
! for this reason, disabling ready and performing stimulus can get the       !
! bus controller out of synchronization with the pod.  Two fault             !
! handlers trap pod timeout conditions that indicate the bus controller      !
! is out of synchronization.  The recover() program is executed to           !
! resynchronize the bus controller and the pod.                              !
!                                                                             !
! TEST PROGRAMS CALLED:                                                       !
!    recover     ()                         The 80286 microprocessor has a!
!                                           bus controller that is totally!
!                                           separate from the pod.  In    !
!                                           some cases the pod can get out!
!                                           of sync with the bus control- !
!                                           ler.  The recover program     !
!                                           resynchronizes the pod and the!
!                                           bus controller.               !
!                                                                          !
!    pia_init()                             Initalization program for the !
!                                           8255.  Sets port A and B to   !
!                                           output with port C to input.  !
!                                                                          !
! GRAPHICS PROGRAMS CALLED:                                                !
!    (none)                                                                !
!                                                                          !
! Local Variables Modified:                                                !
!    devname                                Measurement device            !
!                                                                          !
! Global Variables Modified:                                               !
!    recover_times                          Reset to Zero                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

(continued on the next page)

Figure 4-55: Stimulus Program *(pia_data)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
recover_times = 0

! Let GFI user select which I/O module to use

  if (gfi control) = "yes" then
    devname = gfi device
  else
    devname = "/mod1"
  end if
   print "Stimulus Program PIA_DATA"

! Initialize the PIA and setup the measurement device.

  reset device devname
  pia_init()
  setspace space (getspace space "i/o", size "byte")
  write addr $4002, data  $AA              ! set port B to known value.
  sync device devname, mode "pod"
  sync device "/pod", mode "data"

! Present stimulus to the UUT, read PIA port B register onto data bus.

  arm device devname          ! Start response capture.
      read addr $4002          ! read port B
      write addr $4002, data  $55
      read addr $4002
  readout device devname      ! End response capture.

end pia_data
```

Figure 4-55: Stimulus Program *(pia_data) - continued*

STIMULUS PROGRAM NAME: PIA_DATA
DESCRIPTION: SIZE: 326 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U31-34	PROBE	0003			TRANS		
U31-34	I/O MODULE	0003			TRANS		U21-5
U31-33	PROBE	0004			TRANS		
U31-33	I/O MODULE	0004			TRANS		U21-5
U31-32	PROBE	0003			TRANS		
U31-32	I/O MODULE	0003			TRANS		U21-5
U31-31	PROBE	0004			TRANS		
U31-31	I/O MODULE	0004			TRANS		U21-5
U31-30	PROBE	0003			TRANS		
U31-30	I/O MODULE	0003			TRANS		U21-5
U31-29	PROBE	0004			TRANS		
U31-29	I/O MODULE	0004			TRANS		U21-5
U31-28	PROBE	0003			TRANS		
U31-28	I/O MODULE	0003			TRANS		U21-5
U31-27	PROBE	0004			TRANS		
U31-27	I/O MODULE	0004			TRANS		U21-5

Figure 4-56: Response File *(pia_data)*

```
program pia_leds

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to exercise PIA output signals.                          !
!                                                                           !
! Stimulus programs and response files are used by GFI to backtrace         !
! from a failing node.  The stimulus program must create repeatable UUT     !
! activity and the response file contains the known-good responses for      !
! the outputs in the UUT that are stimulated by the stimulus program.       !
!                                                                           !
! This Stimulus program uses rampdata at the PIA output port addresses      !
! to toggle port B.                                                         !
!                                                                           !
! TEST PROGRAMS CALLED:                                                     !
!    pia_init()                          Initalization program for the      !
!                                        8255.  Sets port A and B to        !
!                                        output with port C to input.       !
!                                                                           !
! GRAPHICS PROGRAMS CALLED:                                                 !
!    (none)                                                                 !
!                                                                           !
! Local Variables Modified:                                                 !
!    devname                             Measurement device                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI user select which I/O module to use

   if (gfi control) = "yes" then
     devname = gfi device
   else
     devname = "/mod1"
   end if
   print "Stimulus Program PIA_LEDS"

! Initialize the PIA port and setup measurent device.

   reset device devname
   execute pia_init()
   setspace space (getspace space "i/o", size "word")
   sync device devname, mode "pod"
   sync device "/pod", mode "data"

! Present stimulus to the UUT

   arm device devname              ! Start response capture.
      rampdata addr $4000, data 0, mask $FF
      rampdata addr $4002, data 0, mask $FF
   readout device devname          ! End response capture

end pia_leds
```

Figure 4-57: Stimulus Program *(pia_leds)*

STIMULUS PROGRAM NAME: PIA_LEDS
DESCRIPTION: SIZE: 1,134 BYTES

Node Signal Src	Learned With	SIG	Response Data Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U31-4	I/O MODULE	EFF7	1	0	TRANS		
U31-3	I/O MODULE	7628	1	0	TRANS		
U31-2	I/O MODULE	790E	1	0	TRANS		
U31-1	I/O MODULE	49CB	1	0	TRANS		
U31-40	I/O MODULE	C04E	1	0	TRANS		
U31-39	I/O MODULE	1D3A	1	0	TRANS		
U31-38	I/O MODULE	A1C7	1	0	TRANS		
U31-37	I/O MODULE	63EB	1	0	TRANS		
U31-18	I/O MODULE	D37A	1	0	TRANS		
U31-19	I/O MODULE	A121	1	0	TRANS		
U31-20	I/O MODULE	6AFA	1	0	TRANS		
U31-21	I/O MODULE	B5FC	1	0	TRANS		
U31-22	I/O MODULE	A71E	1	0	TRANS		
U31-23	I/O MODULE	DAF9	1	0	TRANS		
U31-24	I/O MODULE	23EF	1	0	TRANS		
U31-25	I/O MODULE	2F53	1	0	TRANS		
U46-18	PROBE	D37A	1	0	TRANS		
U46-18	I/O MODULE	D37A	1	0	TRANS		
U46-16	PROBE	A121	1	0	TRANS		
U46-16	I/O MODULE	A121	1	0	TRANS		
U46-14	PROBE	6AFA	1	0	TRANS		
U46-14	I/O MODULE	6AFA	1	0	TRANS		
U46-12	PROBE	B5FC	1	0	TRANS		
U46-12	I/O MODULE	B5FC	1	0	TRANS		
U46-9	PROBE	A71E	1	0	TRANS		
U46-9	I/O MODULE	A71E	1	0	TRANS		
U46-7	PROBE	DAF9	1	0	TRANS		
U46-7	I/O MODULE	DAF9	1	0	TRANS		
U46-5	PROBE	23EF	1	0	TRANS		
U46-5	I/O MODULE	23EF	1	0	TRANS		
U46-3	PROBE	2F53	1	0	TRANS		
U46-3	I/O MODULE	2F53	1	0	TRANS		
U32-18	PROBE	EFF7	1	0	TRANS		
U32-18	I/O MODULE	EFF7	1	0	TRANS		
U32-16	PROBE	7628	1	0	TRANS		
U32-16	I/O MODULE	7628	1	0	TRANS		
U32-14	PROBE	790E	1	0	TRANS		
U32-14	I/O MODULE	790E	1	0	TRANS		
U32-12	PROBE	49CB	1	0	TRANS		
U32-12	I/O MODULE	49CB	1	0	TRANS		
U32-9	PROBE	C04E	1	0	TRANS		
U32-9	I/O MODULE	C04E	1	0	TRANS		

(continued on the next page)

Figure 4-58: Response File *(pia_leds)*

```
U32-7     PROBE       1D3A    1 0     TRANS
U32-7     I/O MODULE  1D3A    1 0     TRANS
U32-5     PROBE       A1C7    1 0     TRANS
U32-5     I/O MODULE  A1C7    1 0     TRANS
U32-3     PROBE       63EB    1 0     TRANS
U32-3     I/O MODULE  63EB    1 0     TRANS
R11-2     PROBE       4596    1       TRANS
R12-2     PROBE       4596    1       TRANS
R13-2     PROBE       4596    1       TRANS
R14-2     PROBE       4596    1       TRANS
R15-2     PROBE       4596    1       TRANS
R16-2     PROBE       4596    1       TRANS
R17-2     PROBE       4596    1       TRANS
R18-2     PROBE       4596    1       TRANS
R19-2     PROBE       4596    1       TRANS
R23-2     PROBE       4596    1       TRANS
R24-2     PROBE       4596    1       TRANS
R25-2     PROBE       4596    1       TRANS
R27-2     PROBE       4596    1       TRANS
R28-2     PROBE       4596    1       TRANS
R29-2     PROBE       4596    1       TRANS
R30-2     PROBE       4596    1       TRANS
```

Figure 4-58: Response File *(pia_leds) - continued*

```
program pia_init

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! INITIALIZATION PROGRAM to set up the PIA.                               !
!                                                                         !
! TEST PROGRAMS CALLED:                                                   !
!    (none)                                                               !
!                                                                         !
! GRAPHICS PROGRAMS CALLED:                                               !
!    (none)                                                               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Set address space

   setspace space (getspace space "i/o", size "byte")

! Initialize the PIA port

   write data $89, addr $4006          ! SET CONTROL REG
   write data $FF, addr $4000          ! CLEAR THE A REG
   write data $FF, addr $4002          ! CLEAR THE B REG

end pia_init
```

Figure 4-59: Initialization Program *(pia_init)*

Summary of Complete Solution for Parallel I/O 4.5.7.

The entire set of programs and files needed to test and GFI troubleshoot the Parallel I/O functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Parallel I/O)

Programs (PROGRAM):

TEST_PIA	Functional Test	Section 4.5.5
PIA_DATA	Stimulus Program	Figure 4-55
PIA_LEDS	Stimulus Program	Figure 4-57
KEY_1	Stimulus Program	Figure 4-47
KEY_2	Stimulus Program	Figure 4-49
KEY_3	Stimulus Program	Figure 4-51
KEY_4	Stimulus Program	Figure 4-53
PIA_INIT	Initialization Program	Figure 4-59

Stimulus Program Responses (RESPONSE):

PIA_DATA	Figure 4-56
PIA_LEDS	Figure 4-58
KEY_1	Figure 4-48
KEY_2	Figure 4-50
KEY_3	Figure 4-52
KEY_4	Figure 4-54

Node List (NODE):

NODELIST	Appendix A

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix B

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

(This page is intentionally blank.)

SERIAL INPUT/OUTPUT FUNCTIONAL BLOCK 4.6.

Introduction to Serial I/O 4.6.1.

The block diagram in Figure 4-60 shows a typical serial I/O port implemented with a UART (universal asynchronous receiver-transmitter) surrounded by its direct support circuitry. For the UART to function properly, all of the support circuitry in Figure 4-1 must function properly.

SIA (serial interface adaptor) chips typically implement all of the UART block and most of the clock and interrupt blocks. On the Demo/Trainer UUT, address decoding and interrupt generation circuits are grouped as separate functional blocks and are described later in Sections 4.11 and 4.13.

Considerations for Testing and Troubleshooting 4.6.2.

Testing

The external I/O lines can be divided into two types:

- Serial lines.
- Handshake and control lines.

Testing the handshake lines is straightforward. The status of input handshake lines can usually be checked by reading a register and testing the appropriate bit. Similarly, output handshake lines can be toggled by setting and clearing a bit in an output register. Testing can be done using the probe or by connecting output lines back to input lines. Some SIA chips need initialization before they respond properly.

Testing the serial input and serial output lines is usually done by connecting the output back to the input. On the Demo/Trainer UUT, this can be done by setting switches. In general, it is

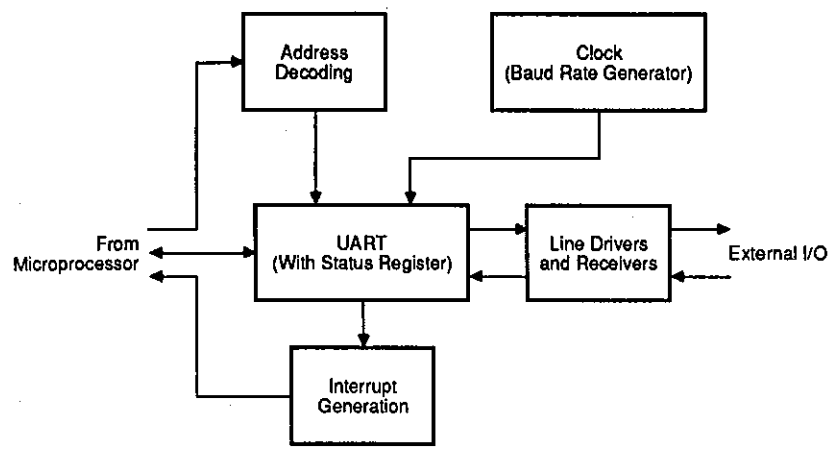


Figure 4-60: Typical Serial I/O Port, With Support Circuitry

preferable to wire a connector to perform the loopback. This allows testing the entire interface, including the connector.

UART chips provide data buffers on their inputs. Therefore, characters can be written to the output side of the UART and the read at the input side. If this technique is used, two limitations should be kept in mind:

- Since the input and output baud rates are usually derived from the same clock, loopback testing will not test for proper baud-rate timing.
- The UART must be initialized with the same transmit and receive baud rate.

One approach to testing the baud rate clock frequency is to set up the transmitter to send seven bits with no parity. Under these conditions, when a null character (00 hex) is sent, the result will be a pulse that is high for eight bit times (start bit and seven data bits). If the probe is connected to a known-frequency clock signal and the start and stop lines are connected to the serial output, the baud rate can be computed. The start line should cause counting to start on the first bit and the stop line should stop the count at the end of the last bit. For example, on the Demo/Trainer UUT, the 8 MHz clock on U1-5 (Figure 4-61) can be probed and the start and stop lines from the clock module can be connected to one of the serial output pins (U13-8 or U12-7). Eight bits at 1200 baud (8/1200 sec) counting 8 MHz the result should be about 53,333 (D055 hex) counts.

The procedures above do not test the interrupt generation block. This circuitry, which is described in detail later in Section 4.13, can be tested by individually enabling the interrupts that are of interest and then stimulating them by exercising the UART. For example, to test the character-received interrupt, perform the following steps:

1. Initialize the interface.
2. Enable the receiver interrupt (usually a bit in a command register).

3. With loopback wired, send a character.

4. Verify that the pod received an interrupt using the *readstatus* TL/1 command. (This assumes that the interrupt stays active until serviced.)

Here are some potential problems in testing serial I/O ports:

- The I/O module may load a crystal oscillator enough to shift the frequency or make it stop oscillating.

- Some SIA chips will not send characters if their handshake lines are in the wrong state.

- If a loopback test cannot be performed on your UUT, you can use the RS-232 port on your 9100A/9105A to test the serial I/O port on the UUT.

Troubleshooting

The central element of a serial I/O port is the UART or SIA chip. If troubleshooting is started by clipping the UART, the problem should be easily isolated. The UART either receives or generates signals from all of the other circuit blocks. If all inputs to the UART are good and all outputs are bad, the UART is bad or its outputs are loaded. If an input is bad, the problem can be traced into the circuitry that generated it. All of this is done automatically in GFI.

The serial input and output can be evaluated by writing a series of characters and counting transitions. The Demo/Trainer UUT stimulus programs for the serial I/O block work this way.

The Demo/Trainer UUT has built-in switches that loop the serial outputs back to the inputs. If GFI troubleshooting is done with the loopback in place, the nodelist must show this connection; if

loopback is done at the connector, the appropriate pins of the connector can simply be shown on the same node.

The probe has a special threshold level for testing RS-232 signals, which is set up with the TL/1 command:

```
threshold device "/probe", mode "rs232"
```

or the operator's keypad command:

```
SET PROBE LOGIC INPUT LEVEL TO RS232.
```

If a part has RS-232-level signals, it should be specified as a probe device in the reflist for the UUT.

The *gfi control* TL/1 command determines when a stimulus program is under GFI (or UFI) control. There are many examples of its use in the stimulus programs that follow. When a program is under GFI (or UFI) control, the *gfi reference* function will return a string describing the device being clipped or the pin being probed. The following TL/1 example shows how the *gfi ref* command could be used in a stimulus program to change the threshold levels if the components to be tested require such a change.

```
if (gfi control) = "yes" then
    str = gfi ref
    if ((str = "U12-14") or (str = "U12-7")) then
        threshold device "/probe", mode "rs232"
    else
        threshold device "/probe", mode "ttl"
    end if
end if
```

Serial I/O Example 4.6.3.

Figure 4-61 shows the serial I/O port on the Demo/Trainer UUT. The DUART (dual universal asynchronous receiver-transmitter), U11, receives serial data input from the keyboard (RXDA/TXDA) and handles bidirectional signal flow with the RS-232 port (RXDB/TXDB). Keyboard input must be at 1200

baud. U12 acts as a level shifter, coupling TTL signal levels on the Demo/Trainer UUT to RS-232 levels at the serial interface; U12 uses a charge pump to shift levels from a +5V source.

The keystroke functional test that follows is not a complete test of the RS-232 circuit. The keyboard receive, port 1 transmit, and port 2 receive lines are not tested between the loopback switch and the connectors. Also, the test assumes that the interrupt functional block is good when testing the INT pin (U11-24).

Keystroke Functional Test 4.6.4.

1. Initialize the Dual UART using the EXEC key with the following command:

```
EXECUTE UUT DEMO PROGRAM RS232_INIT
```

2. Close switches SW4-4, SW4-5 and SW6-4. Now the Transmit line (Txd) is looped back to the receive line (RxD) and transmitting a character on TxD will cause the UART to receive a character on RxD: Then use the SETUP MENU key with the following command to turn off reporting of interrupts:

```
SETUP POD REPORT INTR ACTIVE OFF
```

3. Use the WRITE and READ keys with the following commands to test Port A of the DUART:

```
WRITE DATA 45 TO ADDR 2006
... (ADDR OPTION: I/O BYTE)
READ ADDR 2006 =
... (ADDR OPTION: I/O BYTE)
    The value read should be 45.
```

4. Use the WRITE and READ keys with the following commands to test the Transmit to Receive loopback of Port B of the DUART:

```
WRITE DATA 55 TO ADDR 2016
... (ADDR OPTION: I/O BYTE)
READ ADDR 2016 =
... (ADDR OPTION: I/O BYTE)
     The value read should be 55.
```

You may need to do the READ step up to three times to get the expected value, since the read buffer can be stacked three-deep.

5. Use the WRITE and READ keys with the following commands to test the RTS to CTS loopback of Port B of the DUART:

```
WRITE DATA 0 TO ADDR 201A
... (ADDR OPTION: I/O BYTE)
WRITE DATA FF TO ADDR 201C
... (ADDR OPTION: I/O BYTE)
READ ADDR 201A =
... (ADDR OPTION: I/O BYTE)
     Examine the hexadecimal value to make sure
     bit 1 is a 0.  Bit 0 is the LSB.
WRITE DATA FF TO ADDR 201E
... (ADDR OPTION: I/O BYTE)
READ ADDR 201A =
... (ADDR OPTION: I/O BYTE)
     Examine the hexadecimal value to make sure
     bit 1 is a 1.  Bit 0 is the LSB.
```

Keystroke Functional Test

CONNECTION TABLE

[redacted]		
[redacted]	[redacted]	[redacted]
TEST ACCESS SOCKET	SW6-4 SW4-4 SW4-5	TEST ACCESS SOCKET

STIMULUS AND RESPONSE TABLE FOR DUART PORT A

[redacted]	DATA RECEIVED FROM PORT A (ADDRESS 2006)
	45

STIMULUS AND RESPONSE TABLE FOR DUART PORT B

[redacted]	DATA RECEIVED FROM PORT B (ADDRESS 2016)
	55

STIMULUS AND RESPONSE TABLE FOR TIMER INTERRUPT

[redacted]	BIT 1 LEVEL AT ADDRESS 201A (BIT 0 IS LSB)
	LOW HIGH

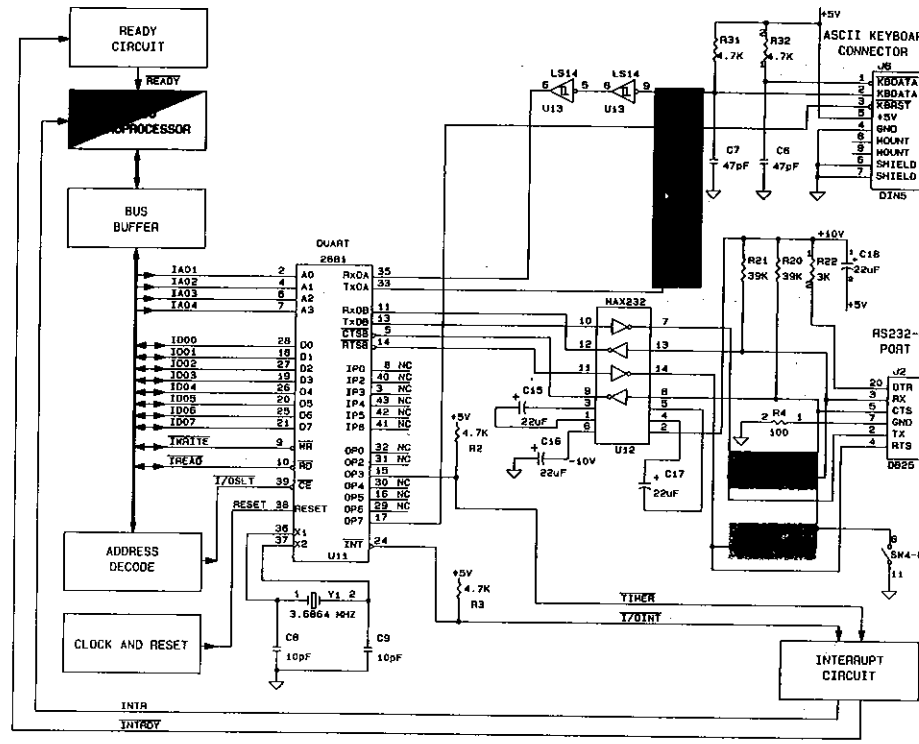


Figure 4-61: Serial I/O Functional Test

Programmed Functional Test 4.6.5.

The *test_rs232* program is the programmed functional test for the Serial I/O functional block. This program also tests for interrupt conditions generated by the Serial I/O circuit.

First, the program initializes the DUART U11 and prompts the test operator to close the loopback switches which connect Port A transmit to Part A receive, connect Port B transmit to Port B receive, and connect Port B Request To Send (RTS) to Port B Clear To Send (CTS).

Next, Port A is checked by transmitting a character and examining the receive buffer for the same character.

And finally, a character is transmitted on Port B which also generates an interrupt condition. Two pod programs called *frc_int* and *rd_cscd* are executed to check proper operation of the interrupt logic. After that, the receive buffer is examined for the same character that was transmitted. This clears the interrupt condition. Then the *frc_int* program is executed again to make sure the interrupt condition has been cleared. A register in the DUART is then checked to see that the RTS/CTS loopback worked properly.

If any of the above operations fail, the *gfi test* command is used to find a failing signal. GFI then takes control and backtraces to the source of the failure.

If a problem is detected in the interrupt circuit, the *tst_intrpt* program (programmed test of the Interrupt Circuit functional block) is executed.

```
program test_rs232

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the SERIAL I/O functional block.                    !
!                                                                        !
! This program tests the SERIAL I/O functional block of the              !
! Demo/Trainer.  The two RS-232 ports are tested by setting three Dip    !
! Switches to loop back the two ports (SW4-4, SW4-5 and SW6-4 loop back !
! ports A and B).  The SERIAL I/O functional block also outputs two      !
! interrupt request signals.  This program also checks the interrupt     !
! circuitry.                                                             !
!                                                                        !
!    frc_int    ()                         POD PROGRAM forces repetitive !
!                                          interrupt acknowledge cycles  !
!                                          and returns first interrupt   !
!                                          vector found on data bus.     !
!                                                                        !
!    rd_cscd    ()                         POD PROGRAM returns the 24 bit!
!                                          interrupt cascade address that!
!                                          was found on the address bus  !
!                                          during the last interrupt     !
!                                          acknowledge cycle.            !
!                                                                        !
!    rd_rearm   ()                         POD PROGRAM returns the most  !
!                                          recent interrupt vector and   !
!                                          rearms the pod to respond to  !
!                                          the next interrupt.           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare
   string q                            ! used to get input from keyboard
   global string rev                   ! Reverse Video escape sequence
   global string norm                  ! Normal Video escape sequence
end declare

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONS                                                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

function sync_buffer( address, data )
   declare numeric address
   declare numeric data

! Synchronize FIFO buffer in DUART.  Write and then read until correct data
! is returned or count has expired.

   write addr address, data data      ! Transmit Data 31 on port A
   wait time $200
   cnt = 0  \ x = 0
   loop until x = data or cnt > 3
      x = read addr address
      cnt = cnt + 1
   end loop
end function
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the SERIAL I/O Functional Block.                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Set interrupt acknowledge cycles on and use the 80286
! pod specific programs rd_rearm(), frc_int() & rd_cscd().

   podsetup 'report intr' "off"
   podsetup 'intr_ack on'          ! Enable Interrupt Ack. cycles
   option = getspace space "i/o", size "byte"
   setspace (option)
   execute check_loop()
   execute rd_rearm()              ! Clear interrupts

! Main part of Test.  Verify DUART port A.

   sync_buffer( $2006, $61 )       ! Synchronize FIFO in DUART for port A
   write addr $2006, data $55      ! Transmit Data 31 on port A
   wait time $200
   if ((read addr $2002) and $F) <> $D then fault 'RS232 Port A failed' \ return
   if (read addr $2006) <> $55 then fault 'RS232 Port A failed' \ return
   write addr $2006, data $55      ! Transmit Data 31 on port A
   wait time $200
   if ((read addr $2002) and $F) <> $D then fault 'RS232 Port A failed' \ return
   if (read addr $2006) <> $55 then fault 'RS232 Port A failed' \ return

! Verify DUART port B and interrupts.

   sync_buffer( $2016, $61 )       ! Synchronize FIFO in DUART for port B
   write addr $201E, data $FF      ! set output port low
   write addr $2016, data $31      ! Transmit Data 31 on port B
   if frc_int() <> $22 then fault 'Interrupt failed' \ return
   if rd_cscd() <> $2016 then fault 'Interrupt failed' \ return
   if (readstatus() and 8) <> 8 then fault 'Interrupt failed' \ return
   if (read addr $2016) <> $31 then fault 'RS232 Port B failed' \ return
   if frc_int() <> $27 then fault 'Interrupt failed' \ return
   write addr $201C, data $FF
   if ((read addr $201A) and 2) <> 0 then fault 'RS232 Port B failed' \ return

end program
```

Stimulus Programs and Responses 4.6.6.

Figure 4-62 is the stimulus program planning diagram for the Serial I/O functional block. The Serial I/O stimulus programs require the test operator to close the loopback switches which loop the transmit lines back to the receive lines and loop the Port B RTS output back to the Port B CTS input.

The *rs232_data* stimulus program outputs data from the DUART onto the data bus. The *rs232_lvl* stimulus program sends a character out the transmit line and then monitors RS232-level signals using the probe with the threshold levels set to "rs232". The *ttl_lvl* stimulus program is the same as *rs232_lvl* except that signals are measured using a level threshold of "ttl".

All the stimulus programs execute *rs232_init* before any measurements are made on the Serial I/O circuitry.

Stimulus Program Planning

PROGRAM: RS232_DATA
EXECUTES RS232_INIT AND READS DATA FROM DUART REGISTERS
MEASUREMENT AT: U11-28,18,27,19,26,20,25,21

PROGRAM: TTL_LVL
EXECUTES RS232_INIT AND EXERCISES RS-232 CIRCUITRY AT TTL LEVELS
MEASUREMENT AT: U11-33,14,24,13,15,17 U12-12,9 U13-6,8

PROGRAM: RS232_LVL
EXECUTES RS232_INIT AND EXERCISES RS-232 CIRCUITRY AT RS-232 LEVELS
MEASUREMENT AT: U12-7,14,1,2,4,6 J2-3,5 R22-2 C15-2 C17-2

INITIALIZATION PROGRAM: RS232_INIT
INITIALIZES THE DUART
MEASUREMENT AT: (NONE)

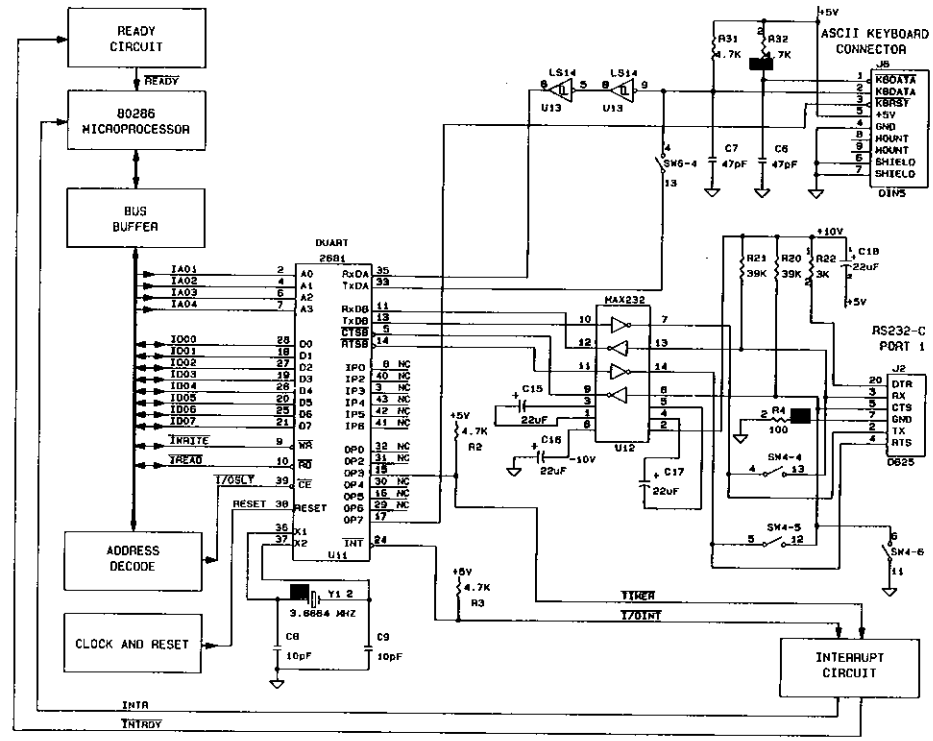


Figure 4-62: Serial I/O Stimulus Program Planning

```
program rs232_data

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM for U11 data lines as outputs.                           !
!                                                                           !
! Stimulus programs and response files are used by GFI to backtrace        !
! from a failing node. The stimulus program must create repeatable UUT     !
! activity and the response file contains the known-good responses for    !
! the outputs in the UUT that are stimulated by the stimulus program.      !
!                                                                           !
! This stimulus program is one of the programs which creates activity      !
! in the kernel area of the UUT. These programs create activity with       !
! or without the ready circuit working properly. Because of this, all      !
! the stimulus programs in the kernel area must disable the READY input    !
! to the pod, then perform the stimulus, then re-enable the READY input    !
! to the pod. The 80286 microprocessor has a separate bus controller;      !
! for this reason, disabling ready and performing stimulus can get the     !
! bus controller out of synchronization with the pod. Two fault            !
! handlers trap pod timeout conditions that indicate the bus controller    !
! is out of synchronization. The recover() program is executed to          !
! resynchronize the bus controller and the pod.                            !
!                                                                           !
! TEST PROGRAMS CALLED:                                                     !
!    rs232_init  ()                        Initialize the RS232 circuit.   !
!                                                                           !
!    recover     ()                        The 80286 microprocessor has a!
!                                          bus controller that is totally!
!                                          separate from the pod.  In    !
!                                          some cases the pod can get out!
!                                          of sync with the bus control- !
!                                          ler.  The recover program     !
!                                          resynchronizes the pod and the!
!                                          bus controller.               !
!                                                                           !
! GRAPHICS PROGRAMS CALLED:                                                 !
!    (none)                                                                 !
!                                                                           !
! Global Variables Modified:                                                !
!    recover_times                         Reset to Zero                   !
!    devname                               Measurement device              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-63: Stimulus Program *(rs232_data)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program RS232_DATA"

! Set addressing mode and setup measurement device.

   reset device devname
   execute rs232_init()
   setspace space (getspace space "i/o", size "byte")
   sync device devname, mode "pod"
   sync device "/pod", mode "data"

! Present stimulus to UUT.

   arm device devname            ! Start response capture.
      read addr $200A
      read addr $201A
      read addr $2012
      read addr $201A
      read addr $2000
   readout device devname        ! End response capture.

end program
```

Figure 4-63: Stimulus Program *(rs232_data) - continued*

STIMULUS PROGRAM NAME: RS232_DATA
DESCRIPTION: SIZE: 318 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U11-18	PROBE	000B		1	0 TRANS		
U11-18	I/O MODULE	000B		1	0 TRANS		
U11-19	PROBE	000E		1	0 TRANS		
U11-19	I/O MODULE	000E		1	0 TRANS		
U11-20	PROBE	000A		1	0 TRANS		
U11-20	I/O MODULE	000A		1	0 TRANS		
U11-21	PROBE	000A		1	0 TRANS		
U11-21	I/O MODULE	000A		1	0 TRANS		
U11-25	PROBE	000A		1	0 TRANS		
U11-25	I/O MODULE	000A		1	0 TRANS		
U11-26	PROBE	001A		1	0 TRANS		
U11-26	I/O MODULE	001A		1	0 TRANS		
U11-27	PROBE	000F		1	0 TRANS		
U11-27	I/O MODULE	000F		1	0 TRANS		
U11-28	PROBE	001B		1	0 TRANS		
U11-28	I/O MODULE	001B		1	0 TRANS		

Figure 4-64: Response File *(rs232_data)*

```
program rs232_lvl

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM for DUART serial circuits at TTL levels.                   !
!                                                                             !
! Stimulus programs and response files are used by GFI to backtrace           !
! from a failing node.  The stimulus program must create repeatable UUT       !
! activity and the response file contains the known-good responses for        !
! the outputs in the UUT that are stimulated by the stimulus program.         !
!                                                                             !
! This stimulus program is one of the programs which creates activity         !
! in the kernel area of the UUT.  These programs create activity with         !
! or without the ready circuit working properly.  Because of this, all        !
! the stimulus programs in the kernel area must disable the READY input       !
! to the pod, then perform the stimulus, then re-enable the READY input       !
! to the pod.  The 80286 microprocessor has a separate bus controller;        !
! for this reason, disabling ready and performing stimulus can get the        !
! bus controller out of synchronization with the pod.  Two fault              !
! handlers trap pod timeout conditions that indicate the bus controller       !
! is out of synchronization.  The recover() program is executed to            !
! resynchronize the bus controller and the pod.                               !
!                                                                             !
! TEST PROGRAMS CALLED:                                                       !
!    rs232_init  ()                       Initialize the RS232 circuit.       !
!                                                                             !
!    check_loop  ()                       Check that loop-back switches       !
!                                         are closed.  Prompt if the          !
!                                         switches are not closed.            !
!                                                                             !
! GRAPHICS PROGRAMS CALLED:                                                   !
!    (none)                                                                   !
!                                                                             !
! Local Variables Modified:                                                   !
!    q                                    String to accept keypad input.      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  declare string q

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the measurement device.

  if (gfi control) = "yes" then
     devname = gfi device
  else
     devname = "/mod1"
  end if
  print "Stimulus Program RS232_LVL"
```

(continued on the next page)

Figure 4-65: Stimulus Program *(rs232_lvl)*

```
! Set addressing mode and setup measurement device.

   reset device devname
   execute rs232_init()
   setspace space (getspace space "i/o", size "byte")
   sync device "/probe", mode "freerun"
   threshold device "/probe", level "rs232"

   execute check_loop()          ! check if the loop back switches are set.

! Present stimulus to UUT.

   arm device devname            ! Start response capture.
      write addr $2006, data $55   !    Txd port A
      write addr $2006, data  $D   !    Txd port A
      write addr $2016, data $55   !    Txd port B
      write addr $2016, data  $D   !    Txd port B
   readout device devname        ! End response capture.

end program
```

Figure 4-65: Stimulus Program *(rs232_lvl) - continued*

STIMULUS PROGRAM NAME: RS232_LVL
DESCRIPTION: SIZE: 249 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U12-7	PROBE		1	0	TRANS	8	
U12-14	PROBE		1		TRANS	0	
J2-3	PROBE		1	0	TRANS	8	
J2-5	PROBE		1		TRANS	0	
R22-2	PROBE		1		TRANS	0	
U12-1	PROBE		1		TRANS		
U12-2	PROBE		1		TRANS		
C15-2	PROBE		1X		TRANS		
U12-4	PROBE		1X		TRANS		
C17-2	PROBE		1X0		TRANS		
U12-6	PROBE		X		TRANS		

Figure 4-66: Response File *(rs232_lvl)*

```
program ttl_lvl

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM for DUART serial circuits at TTL levels.                 !
!                                                                           !
! Stimulus programs and response files are used by GFI to back-trace        !
! from a failing node.  The stimulus program must create repeatable UUT     !
! activity and the response file contains the known-good responses for      !
! the outputs in the UUT that are stimulated by the stimulus program.       !
!                                                                           !
! TEST PROGRAMS CALLED:                                                     !
!    rs232_init ()                        Initialize the RS232 circuit.     !
!                                                                           !
!    check_loop ()                        Check that loop-back switches     !
!                                         are closed.  Prompt if the        !
!                                         switches are not closed.          !
!                                                                           !
! GRAPHICS PROGRAMS CALLED:                                                 !
!    (none)                                                                 !
!                                                                           !
! Local Variables Modified:                                                 !
!    q                                    String to accept keypad input.    !
!    devname                              Measurement device                !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare string q

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program TTL_LVL"
```

(continued on the next page)

Figure 4-67: Stimulus Program *(ttl_lvl)*

```
! Set addressing mode and setup measurement device.

   reset device devname
   execute rs232_init()
   setspace space (getspace space "i/o", size "byte")
   sync device "/probe", mode "pod"
   sync device "/pod", mode "data"
   threshold device "/probe", level "ttl"

   execute check_loop()            ! Check if loop back switches are closed.

! Present stimulus to UUT.

   arm device devname              ! Start response capture.
      write addr $2006, data $55   !    Txd port A
      write addr $2006, data  $D   !    Txd port A
      write addr $2016, data $55   !    Txd port B
      write addr $2016, data  $D   !    Txd port B
      write addr $201C, data $FF   !
      write addr $201E, data $FF   ! Pulse timer interrupt.
   readout device devname          ! End response capture.

end program
```

Figure 4-67: Stimulus Program *(ttl_lvl) - continued*

```
STIMULUS PROGRAM NAME: TTL_LVL
DESCRIPTION:                                          SIZE:          368 BYTES

                  ------------------- Response Data --------------------
   Node            Learned                Async Clk Counter                    Priority
Signal Src          With          SIG       LVL  LVL  Mode     Counter Range     Pin

U11-13          PROBE                       1  0       TRANS    8
U11-14          PROBE                       1  0       TRANS    1
U11-33          PROBE                       1  0       TRANS    8
U11-33          I/O MODULE                  1  0       TRANS    8
U11-15          PROBE                       1  0       TRANS    1
U11-15          I/O MODULE                  1  0       TRANS    1
U11-17          PROBE                       1  0       TRANS    1
U11-24          PROBE                       1  0       TRANS    0
U11-24          I/O MODULE                  1  0       TRANS    0
U12-12          PROBE                       1  0       TRANS    8
U12-9           PROBE                       1  0       TRANS    1
U13-6           PROBE                       1  0       TRANS    8
U13-6           I/O MODULE                  1  0       TRANS    8
U13-8           I/O MODULE                  1  0       TRANS    8
```

Figure 4-68: Response File *(ttl_lvl)*

```
program rs232_init

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! INITIALIZATION PROGRAM for SERIAL I/O functional block.                    !
!                                                                            !
! TEST PROGRAMS CALLED:                                                      !
!     (none)                                                                 !
!                                                                            !
! GRAPHICS PROGRAMS CALLED:                                                  !
!     (none)                                                                 !
!                                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    setspace space (getspace space "i/o", size "byte")

    write addr $2004, data $15    ! Cmnd Reg A: reset Rxd
    write addr $2004, data $25    ! Cmnd Reg A: reset Txd
    write addr $2004, data $35    ! Cmnd Reg A: reset Errors
    write addr $2004, data $45    ! Cmnd Reg B: reset Rxd
    write addr $2004, data $55    ! Cmnd Reg B: reset Txd
    write addr $2014, data $15    ! Cmnd Reg A: reset Rxd
    write addr $2014, data $25    ! Cmnd Reg A: reset Txd
    write addr $2014, data $35    ! Cmnd Reg A: reset Errors
    write addr $2014, data $45    ! Cmnd Reg B: reset Rxd
    write addr $2014, data $55    ! Cmnd Reg B: reset Txd
    write addr $2000, data $13    ! Mode register 1A
    write addr $2000, data  7     ! Mode register 2A
    write addr $2010, data $13    ! Mode register 1B
    write addr $2010, data  7     ! Mode register 2B
    write addr $2002, data $66    ! Clock select register A
    write addr $2012, data $BB    ! Clock select register B
    write addr $200A, data $20    ! Interrupts for port B
    read addr $2002               ! Read Status Reg A
    read addr $2000               ! Read Command Reg A

end program
```

Figure 4-69: Initialization Program *(rs232_init)*

Summary of Complete Solution for Serial I/O 4.6.7.

The entire set of programs and files needed to test and GFI troubleshoot the Serial I/O functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Serial I/O)

Programs (PROGRAM):

TEST_RS232	Functional Test	Section 4.6.5
RS232_DATA	Stimulus Program	Figure 4-63
RS232_LVL	Stimulus Program	Figure 4-65
TTL_LVL	Stimulus Program	Figure 4-67
FREQUENCY	Stimulus Program	Figure 4-117
LEVELS	Stimulus Program	Figure 4-92
RS232_INIT	Initialization Program	Figure 4-69

Stimulus Program Responses (RESPONSE):

RS232_DATA	Figure 4-64
RS232_LVL	Figure 4-66
TTL_LVL	Figure 4-68
FREQUENCY	Figure 4-118
LEVELS	Figure 4-93

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

VIDEO OUTPUT FUNCTIONAL BLOCK 4.7.

Introduction to Video Output Circuits 4.7.1.

Video output circuits are part of larger video display circuits. In general, video display circuits can be divided into two basic classes: video display controllers and intelligent command-oriented display systems, which are a superset of video display controllers. In this manual, we will limit our discussion to video display controllers.

Figure 4-70 is a block diagram of a typical, complete video display controller, of which video output is one functional block. On the Demo/Trainer UUT, address decoding is partitioned as a separate functional block and is described later in Sections 4.11. Often, much of the video control circuitry is performed by a VDC (video display controller) chip. On the Demo/Trainer UUT, most of the video output block is implemented with a single LSI chip.

The video output block typically performs all or some of the following functions:

- Converts video RAM character or dot graphics signals (typically on a bus) to higher-speed (typically serial) pixel outputs that drive the monitor. This is usually done with shift registers.
- Modifies the meaning of video RAM color-data outputs according to a color look-up table or palette RAM.
- Converts the pixel output to analog or digital signals compatible with the monitor.

Considerations for Testing and Troubleshooting 4.7.2.

The Video Output functional block simply processes information presented to it by the Video Control and Video RAM functional blocks. All three video blocks can be considered good if the

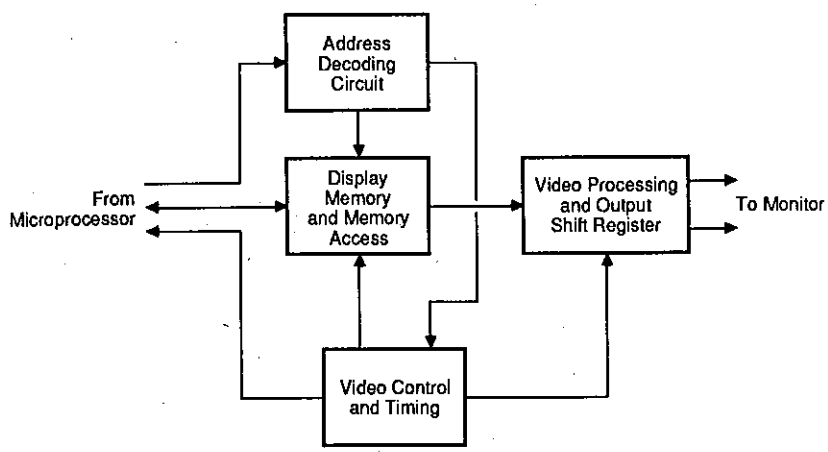


Figure 4-70: Typical Video Controller Circuit

final outputs of the Video Output functional block are good. Because of this, the Video Output functional block is tested first.

While a generalized approach to testing Video Control functional blocks is feasible, testing Video Output and Video RAM functional blocks is strongly dependent on the design of the UUT.

The general approach for testing video circuits is to initialize video RAM and any other RAM sections so that some regular pattern will occur each frame. When this is done for each mode, there should be a way to capture stable signatures on the outputs.

To test video output:

1. Initialize the video control circuit.
2. Initialize the video RAM with blinking disabled.

For horizontal sync and vertical sync:

3. Probe the horizontal sync and vertical sync outputs.
4. Compare all frequencies to those from a known-good UUT.

For video outputs:

3. Connect the clock module's external CLOCK, START, and STOP lines.
4. Compare signatures of TTL-level video outputs to those from a known-good UUT.
5. You can check the level history of any non-TTL-level video outputs to verify that they are toggling.

Connecting the Start and Stop lines to the vertical sync line will usually work. The Clock line should be connected to the high-speed clock that drives the video output shift registers.

Video outputs are sometimes high-speed analog signals. Fortunately, any digital-to-analog conversion is usually done at the last step before the monitor. By measuring the digital signals that drive digital-to-analog converters, most of the circuit can be tested with the 9100A/9105A.

Furthermore, many of the monitors for personal computers accept TTL-level signals. Video cards that put out such TTL-level signals can be checked by the 9100A/9105A at these TTL-level video outputs.

Choose your measurement device to suit the data rate of the signals you are measuring. If the Video Output signals exceed the maximum data rate of the I/O modules (10 MHz), the probe should be used.

Testing should be started in the mode that tests as much of the video display circuitry as possible. In a color graphics circuit, this might be the highest resolution mode with the most colors. Simple tests in other modes can then be used to cover circuitry not tested with the more extensive test.

When selecting the Start and Stop signals for signature analysis, connect to the slowest repetitive signal, relative to the circuitry being tested. This will usually be the vertical sync signal.

To test blinking cursors, it may be easiest simply to probe an internal line to make sure it is blinking rather than run a test program. Other similar modes may also be faster to test with the probe.

Video Output Circuit Example 4.7.3.

The Video Output functional block, shown in Figure 4-71, consists of the 2675 attributes controller chip (U78) and associated circuitry. The 2675 contains a programmable dot

clock divider to generate a character clock, a high-speed shift register to convert parallel pixel data into a serial stream, latches and logic to apply visual attributes (e.g. colors) to the resulting display, and logic to display a cursor on the monitor.

Associated circuitry includes latches U87 and U76, which clock in display information provided by the character PROM, and Q1 and Q2, which boost the video signal before it is mixed with the horizontal and vertical sync signals at the monitor to be connected at J3.

The circuitry from the Video Control functional block up to the 2675 attributes controller chip (U78) clocks video data in character format. This means that the code for a character and the attributes for that character are clocked toward the 2675 chip. The attributes controller converts the parallel character information to pixel data.

The circuitry after U78 should be initialized without blinking characters in the video screen, otherwise the pixel stream will change when the characters blink. However, the circuitry between the video control and U78 may contain blinking characters, since the blinking characters are determined by an attribute bit which is stable.

Keystroke Functional Test 4.7.4.

Before testing any part of the video display circuitry, the video controller and video RAM must be initialized. The TL/1 programs *video_init*, *video_fill*, and *video_fil2* are used for initialization of the Demo/Trainer UUT video circuitry. Figure 4-79 shows the *video_init* program, which contains a sequence of *write* commands needed to initialize the Video Control functional block. Figures 4-80 and 4-81 show the *video_fill* and *video_fil2* programs, which write blocks of data to video RAM.

1. Use the EXEC key with the following commands to initialize the video circuit and to fill the video RAM with a test pattern.

```
EXECUTE UUT DEMO PROGRAM VIDEO_INIT
EXECUTE UUT DEMO PROGRAM VIDEO_FIL1
```

2. Connect the external control lines of the clock module as follows:

   Clock to 16MHZ (U25-9)
   Start to VSYNC (U72-18)
   Stop to VSYNC (U72-18)
   Enable to BLANK (U72-17)

3. Use the SYNC and PROBE keys with the following commands to measure the node response for the video output signals (TTV1, TTLV2, and VIDEO). The pins to be probed and the correct responses are shown in the response table of Figure 4-71.

```
SYNC PROBE TO EXT MOD ENABLE LOW CLOCK ↓ ...
... START ↓ STOP ↑
ARM  PROBE FOR CAPTURE USING SYNC
SHOW PROBE CAPTURED RESPONSES <see ...
... response table>
```

4. Use the PROBE and SOFT KEYS keys with the following command to measure frequency of the video synchronization signals. The results for each sync signal (HSYNC and VSYNC) are shown in the response table of Figure 4-71.

```
FREQ AT PROBE
```

(This page is intentionally blank.)

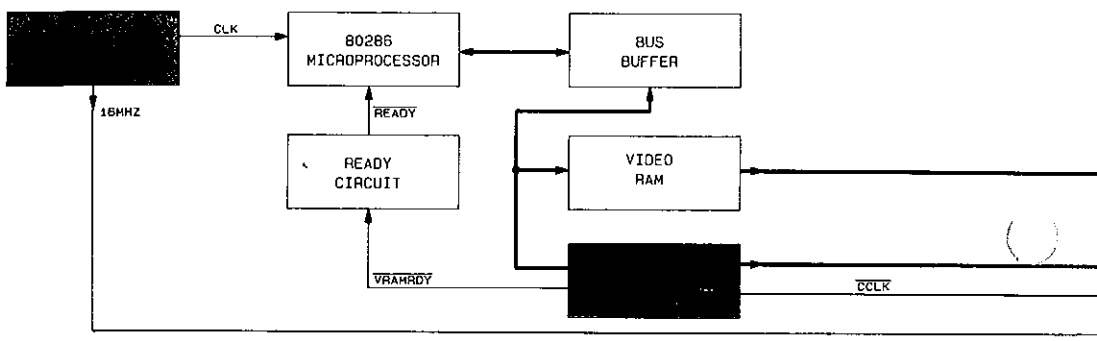
Keystroke Functional Test

CONNECTION TABLE

	MEASUREMENT CONTROL	MEASUREMENT
(NONE)	CLOCK MODULE	PROBE
	CLOCK U25-9 START U72-18 STOP U72-18 ENABLE U72-17	U78 J3

RESPONSE TABLE

SIGNAL	PART PIN	MEASUREMENT
HSYNC	-8	16.7 TO 16.8 KHz
VSYNC	-9	59 TO 61 Hz
VIDEO	J3-7	1X0 (ASYNC LEVEL)
TTLV1	U78-28	B013 (SIG)
TTLV2	-29	E4A7 (SIG)



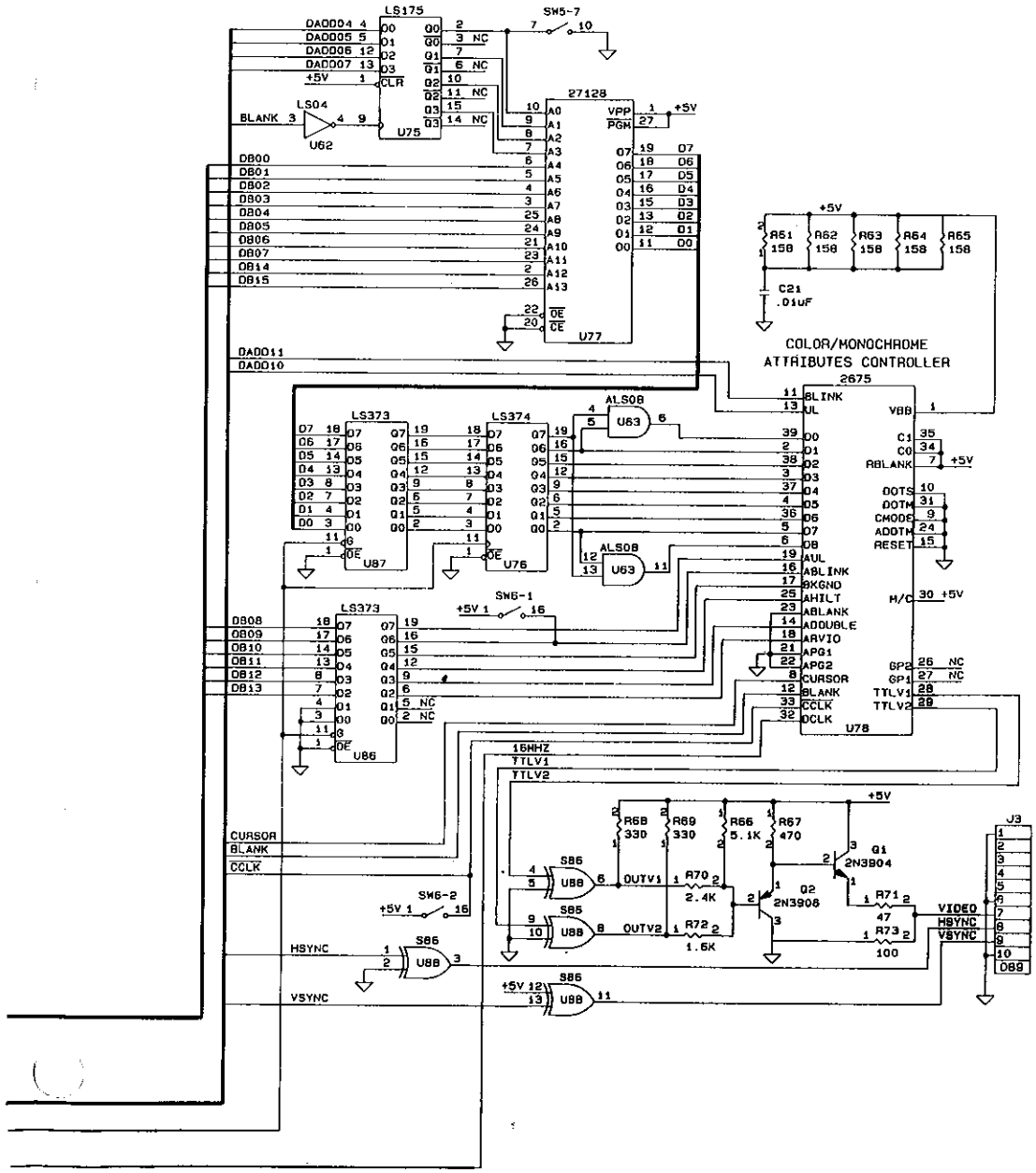


Figure 4-71: Video Output Functional Test

Programmed Functional Test 4.7.5.

The *test_video* program is the programmed functional test for the Video Output functional block. This program uses the *gfi test* command and the probe to measure the output of the video circuit.

If the video outputs fail, the program executes programmed functional tests for the Video Control functional block and the Video RAM functional block. If either of these functional tests fails, GFI will take control and begin backtracing. If neither test fails, the problem is in the Video Output functional block and the *test_video* program passes control to GFI to start backtracing from the video outputs that failed.

```
program test_video

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the VIDEO functional block                          !
!                                                                        !
! This program tests the VIDEO functional block of the Demo/Trainer.     !
! The video test uses the gfi test command to run stimulus programs and !
! to check the outputs of the Video circuit against the stimulus program!
! response files.  The gfi test command returns a passes status if all  !
! the measured results from running the stimulus programs match the      !
! response files.  Otherwise the gfi test command returns a fails        !
! status.                                                                !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Setup and initialization.

   connect clear "yes"
   podsetup 'enable ~ready' "on"
   print "\nl\nl"

! Main part of Test.

   if gfi test "J3-8" fails then fault video_scan \ return
   if gfi test "J3-9" fails then fault video_scan \ return

   if gfi test "U78-11" fails then fault video_scan \ return
   if gfi test "U78-28" fails then fault video_output \ return
   if gfi test "U78-29" fails then fault video_output \ return
   if gfi test "J3-7" fails then fault video_output \ return

end program
```

Stimulus Programs and Responses 4.7.6.

Figure 4-72 is the stimulus program planning diagram for the Video Output functional block. The *video_freq* stimulus program initializes the video registers and then measures frequency. The *video_scan* stimulus program initializes video RAM with blinking characters by executing *video_fill*. The *video_out* stimulus program initializes video RAM without any blinking characters by executing *video_fil2*. Not having blinking characters results in stable signatures in the circuitry between U78 and the video output connector.

All the stimulus programs execute *video_init* before any measurements are made on the video circuitry.

Stimulus Program Planning

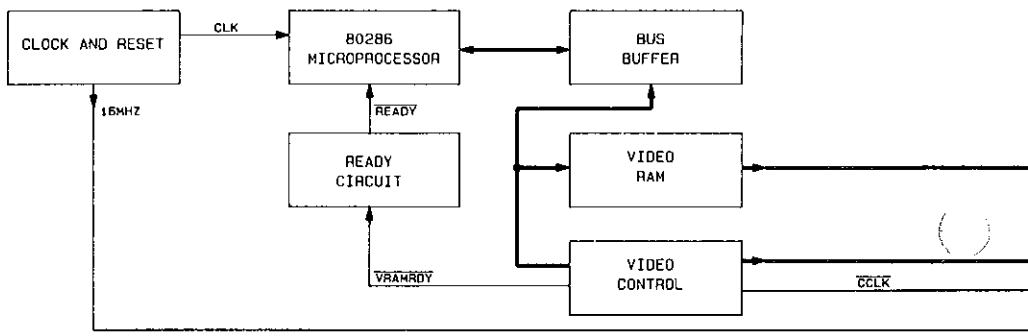
PROGRAM: VIDEO_FREQ
EXECUTES VIDEO_INIT AND MEASURES FREQUENCY
MEASUREMENT AT: U78-33 U88-3,11 U62-4

PROGRAM: VIDEO_SCAN
EXECUTES VIDEO_INIT, VIDEO_FIL1, AND MEASURES ALL CIRCUITRY WHERE DATA IS CLOCKED THROUGH BY CHARACTERS
MEASUREMENT AT: **U75-2,3,7,6,10,11,15,14** **U77-11,12,13,15,16,17,18,19** **U87-2,5,6,9,12,15,16,19** **U76-2,5,6,9,12,15,16,19** **U63-6,11** **U86-6,9,12,15,16,19**

INITIALIZATION PROGRAM: VIDEO_FIL1
INITIALIZES VIDEO RAM WITH BLINKING CHARACTERS
MEASUREMENT AT: (NONE)

INITIALIZATION PROGRAM: VIDEO_INIT
INITIALIZES VIDEO REGISTERS TO STANDARD OPERATING MODE
MEASUREMENT AT: (NONE)

INITIALIZATION PROGRAM: VIDEO_FIL2
INITIALIZES VIDEO RAM WITHOUT BLINKING CHARACTERS
MEASUREMENT AT: (NONE)



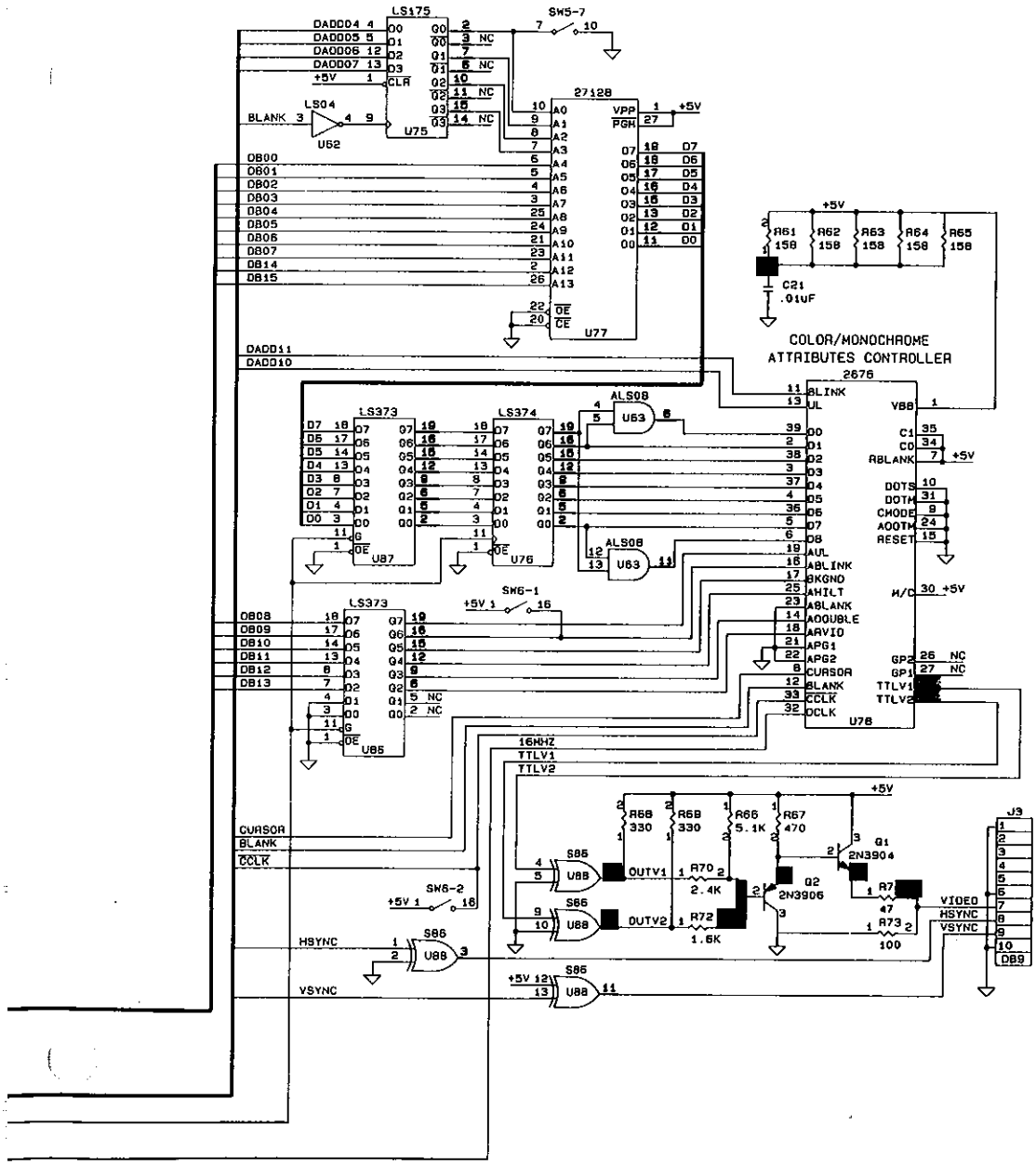


Figure 4-72: Video Output Stimulus Program Planning

```
program video_freq

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to measure frequency in video circuit.                    !
!                                                                            !
! Stimulus programs and response files are used by GFI to backtrace          !
! from a failing node.  The stimulus program must create repeatable UUT      !
! activity and the response file contains the known-good responses for       !
! the outputs in the UUT that are stimulated by the stimulus program.        !
!                                                                            !
! TEST PROGRAMS CALLED:                                                      !
!    video_init ()                                 Initialize video          !
!                                                                            !
! GRAPHICS PROGRAMS CALLED:                                                  !
!    (none)                                                                  !
!                                                                            !
! Local Variables Modified:                                                  !
!     devname                                      Measurement device        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine the measurement device

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/probe"
   end if
   print "\1B[2J"
   print "Stimulus Program VIDEO_FREQ"

! Initialize and Setup desired measurement mode

   reset device devname
   execute video_init()
   counter device devname, mode "freq"

! No stimulus is applied; response is frequency

   arm device devname             ! Start response capture
   readout device devname         ! End response capture

end program
```

Figure 4-73: Stimulus Program *(video_freq)*

STIMULUS PROGRAM NAME: VIDEO_FREQ
DESCRIPTION: SIZE: 345 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U72-17	PROBE		1	0	FREQ	14300-14500	
U72-17	I/O MODULE		1	0	FREQ	14300-14500	
U72-18	PROBE		1	0	FREQ	59-61	
U72-18	I/O MODULE		1	0	FREQ	59-61	
U72-19	PROBE		1	0	FREQ	16700-16800	
U72-19	I/O MODULE		1	0	FREQ	16700-16800	
U78-33	PROBE		1	0	FREQ	1770000-1780000	
U78-33	I/O MODULE		1	0	FREQ	1770000-1780000	
U88-3	PROBE		1	0	FREQ	16700-16800	
U88-11	PROBE		1	0	FREQ	59-61	
U70-11	PROBE		1	0	FREQ	1770000-1780000	
U70-11	I/O MODULE		1	0	FREQ	1770000-1780000	
U62-4	I/O MODULE		1	0	FREQ	14300-14500	

Figure 4-74: Response File *(video_freq)*

```
program video_out

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM measures character scan circuitry from U78 to output.!
!                                                                       !
! Stimulus programs and response files are used by GFI to backtrace     !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    video_init ()                      Initialize video ciruit.        !
!                                                                       !
!    video_fil2 ()                      Initialize data in video RAM    !
!                                       with no blinking characters     !
!                                                                       !
!    check_meas (device, start, stop, clock, enable)                    !
!                                       Checks to see if the measure-   !
!                                       ment is complete using the      !
!                                       TL/1 checkstatus command.  If   !
!                                       the measurement times out then!
!                                       redisplay connect locations.    !
!                                                                       !
! GRAPHICS PROGRAMS CALLED:                                             !
!    (none)                                                             !
!                                                                       !
! Local Variables Modified:                                             !
!    done                               returned from check_meas()      !
!    devname                            Measurement device              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare numeric done = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/probe"
   end if
   print "\1B[2J"
   print "Stimulus Program VIDEO_OUT"
```


(continued on the next page)

Figure 4-75: Stimulus Program *(video_out)*

```
! Initialize and Prompt user to connect external lines

   execute video_init()
   execute video_fil2()
   connect device devname, start "U88-13", stop "U88-13", clock "U25-9", common "gnd"

! Setup desired measurement modes.

   reset device devname
   sync device devname, mode "ext"
   enable device devname, mode "always"
   edge device devname, start "-", stop "+", clock "-"
   old_cal = getoffset device devname
   setoffset device devname, offset (1000000 + 40)

! Present stimulus to UUT.

   loop until done = 1
      arm device devname
         done = check_meas(devname, "U88-13", "U88-13", "U25-9", "*")
      readout device devname
   end loop

   setoffset device devname, offset old_cal
end program
```

Figure 4-75: Stimulus Program *(video_out) - continued*

```
STIMULUS PROGRAM NAME: VIDEO_OUT
DESCRIPTION:                                        SIZE:          200 BYTES

                    ------------------- Response Data --------------------
   Node              Learned                Async Clk Counter                  Priority
Signal Src           With        SIG        LVL  LVL  Mode     Counter Range     Pin

U78-28               PROBE       B013       1    0    TRANS    4431
U78-29               PROBE       E4A7       1    0    TRANS    6359
U88-6                PROBE                  1    0    TRANS    4431
U88-8                PROBE                  1    0    TRANS    6359
R72-2                PROBE                  1X0       TRANS
Q2-1                 PROBE                  1X        TRANS
Q1-1                 PROBE                  1X0       TRANS
R71-2                PROBE                  1X0       TRANS
```

Figure 4-76: Response File *(video_out)*

```
program video_scan

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to measure character scan circuitry from U72 to U78. !
!                                                                         !
! Stimulus programs and response files are used by GFI to back-trace     !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                         !
! TEST PROGRAMS CALLED:                                                   !
!    video_init ()                        Initialize video ciruit.       !
!                                                                         !
!    video_fill ()                        Initialize data in video RAM   !
!                                                                         !
!    check_meas (device, start, stop, clock, enable)                     !
!                                         Checks to see if the measure- !
!                                         ment is complete using the     !
!                                         TL/1 checkstatus command.  If !
!                                         the measurement times out then!
!                                         redisplay connect locations.  !
!                                                                         !
! GRAPHICS PROGRAMS CALLED:                                               !
!    (none)                                                               !
!                                                                         !
! Local Variables Modified:                                               !
!    done                                  returned from check_meas()    !
!    devname                                  Measurement device         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare numeric done = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FAULT HANDLERS:                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Main part of STIMULUS PROGRAM                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0
```

(continued on the next page)

Figure 4-77: Stimulus Program *(video_scan)*

```
! Let GFI determine measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
      measure_ref = gfi ref
   else
      devname = "/mod1"
      measure_ref = "U72"
   end if
   print "Stimulus Program VIDEO_SCAN"

! Initialize and Prompt user to connect external lines

   execute video_init()
   execute video_fill()
   connect device devname, start "U88-13", stop "U88-13", enable "U78-12",
                           clock "U78-33", common "gnd"

! Setup desired mesurement modes.

   reset device devname
   sync device devname, mode "ext"
   enable device devname, mode "low"
   edge device devname, start "-", stop "+", clock "-"

! Present stimulus to the UUT.
! The blink signal node (U72-23 to U78-11) has a signature of 0000 50% of the time
! and the signature in BLINK_SIG the rest of the time.  If U72 or U78-11 is being
! tested, make sure both a zero and the signature in BLINK_SIG are measured
! on the node.  The signature that gfi will evaluate is the signature in the
! variable BLINK_SIG.

   done = 0   \  done2 = 0
   cnt = 0  \  blink = 0
   loop until done = 1 and done2 = 1 or cnt > 12
      arm device devname
         done = check_meas(devname, "U88-13", "U88-13", "U78-33", "U78-12")
         if done = 1 then if checkstatus(devname) <> $F then done2 = 1
      readout device devname
      if measure_ref = "U78-11" then
         if (sig device devname, pin 11)=0 then blink = 1
         if (sig device devname, pin 11)=BLINK_SIG and blink=1 then done2=1
      else if measure_ref = "U72" then
         if (sig device "U72", pin 23)=0 then blink = 1
         if (sig device "U72", pin 23)=BLINK_SIG and blink = 1 then done2 = 1
      else
         done2 = 1              ! Don't loop if not U72 or U78-11
      end if
      cnt = cnt + 1
   end loop

end program
```

Figure 4-77: Stimulus Program *(video_scan) - continued*

STIMULUS PROGRAM NAME: VIDEO_SCAN
DESCRIPTION: SIZE: 1,710 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U74-9	I/O MODULE	4155	1	0	TRANS		
U74-10	I/O MODULE	3F33	1	0	TRANS		
U74-11	I/O MODULE	A65A	1	0	TRANS		
U74-13	I/O MODULE	9024	1	0	TRANS		
U74-14	I/O MODULE	DE6D	1	0	TRANS		
U74-15	I/O MODULE	D6FA	1	0	TRANS		
U74-16	I/O MODULE	7AC3	1	0	TRANS		
U74-17	I/O MODULE	0477	1	0	TRANS		
U85-9	I/O MODULE	A814	1	0	TRANS		
U85-10	I/O MODULE	C26B	1	0	TRANS		
U85-11	I/O MODULE	D909	1	0	TRANS		
U85-13	I/O MODULE	5FAA	1	0	TRANS		
U85-14	I/O MODULE	5925	1	0	TRANS		
U85-15	I/O MODULE	610D	1	0	TRANS		
U85-16	I/O MODULE	B8AB	1	0	TRANS		
U85-17	I/O MODULE	ADD3	1	0	TRANS		
U84-12	I/O MODULE	4155	1	0	TRANS		
U84-9	I/O MODULE	3F33	1	0	TRANS		
U84-7	I/O MODULE	A65A	1	0	TRANS		
U84-4	I/O MODULE	9024	1	0	TRANS		
U83-12	I/O MODULE	DE6D	1	0	TRANS		
U83-9	I/O MODULE	D6FA	1	0	TRANS		
U83-7	I/O MODULE	7AC3	1	0	TRANS		
U83-4	I/O MODULE	0477	1	0	TRANS		
U73-12	I/O MODULE	E941	1	0	TRANS		
U73-9	I/O MODULE	88B8	1	0	TRANS		
U73-7	I/O MODULE	60B0	1	0	TRANS		
U72-34	I/O MODULE	4155	1	0	TRANS		
U72-33	I/O MODULE	3F33	1	0	TRANS		
U72-32	I/O MODULE	A65A	1	0	TRANS		
U72-31	I/O MODULE	9024	1	0	TRANS		
U72-30	I/O MODULE	DE6D	1	0	TRANS		
U72-29	I/O MODULE	D6FA	1	0	TRANS		
U72-28	I/O MODULE	7AC3	1	0	TRANS		
U72-27	I/O MODULE	0477	1	0	TRANS		
U72-26	I/O MODULE	E941	1	0	TRANS		
U72-25	I/O MODULE	88B8	1	0	TRANS		
U72-24	PROBE	60B0	1	0	TRANS		
U72-24	I/O MODULE	60B0	1	0	TRANS		
U72-23	PROBE	D869	1	0	TRANS		
U72-23	I/O MODULE	D869	1	0	TRANS		
U72-7	PROBE	0000		0	TRANS		
U72-7	I/O MODULE	0000		0	TRANS		
U75-2	I/O MODULE	AC4E	1	0	TRANS		

(continued on the next page)

Figure 4-78: Response File *(video_scan)*

```
U75-3       I/O MODULE  6FB1    1 0     TRANS
U75-7       I/O MODULE  9B47    1 0     TRANS
U75-6       I/O MODULE  58B8    1 0     TRANS
U75-10      I/O MODULE  762E    1 0     TRANS
U75-11      I/O MODULE  B5D1    1 0     TRANS
U75-15      I/O MODULE  2C30    1 0     TRANS
U75-14      I/O MODULE  EFCF    1 0     TRANS
U77-11      I/O MODULE  7B80    1 0     TRANS
U77-12      I/O MODULE  8FE6    1 0     TRANS
U77-13      I/O MODULE  ADD1    1 0     TRANS
U77-15      I/O MODULE  EB37    1 0     TRANS
U77-16      I/O MODULE  FFE7    1 0     TRANS
U77-17      I/O MODULE  B708    1 0     TRANS
U77-18      I/O MODULE  55C3    1 0     TRANS
U77-19      I/O MODULE  B00D    1 0     TRANS
U87-2       I/O MODULE  7B80    1 0     TRANS
U87-5       I/O MODULE  8FE6    1 0     TRANS
U87-6       I/O MODULE  ADD1    1 0     TRANS
U87-9       I/O MODULE  EB37    1 0     TRANS
U87-12      I/O MODULE  FFE7    1 0     TRANS
U87-15      I/O MODULE  B708    1 0     TRANS
U87-16      I/O MODULE  55C3    1 0     TRANS
U87-19      I/O MODULE  B00D    1 0     TRANS
U76-2       PROBE       1ADB    1 0     TRANS
U76-2       I/O MODULE  1ADB    1 0     TRANS
U76-5       PROBE       444F    1 0     TRANS
U76-5       I/O MODULE  444F    1 0     TRANS
U76-6       PROBE       D65A    1 0     TRANS
U76-6       I/O MODULE  D65A    1 0     TRANS
U76-9       PROBE       4366    1 0     TRANS
U76-9       I/O MODULE  4366    1 0     TRANS
U76-12      PROBE       49EA    1 0     TRANS
U76-12      I/O MODULE  49EA    1 0     TRANS
U76-15      PROBE       4DDC    1 0     TRANS
U76-15      I/O MODULE  4DDC    1 0     TRANS
U76-16      PROBE       5B18    1 0     TRANS
U76-16      I/O MODULE  5B18    1 0     TRANS
U76-19      I/O MODULE  3EF2    1 0     TRANS
U63-11      PROBE       0C5B    1 0     TRANS
U63-11      I/O MODULE  0C5B    1 0     TRANS
U63-6       PROBE       66D3    1 0     TRANS
U63-6       I/O MODULE  66D3    1 0     TRANS
U86-6       PROBE       610D    1 0     TRANS
U86-6       I/O MODULE  610D    1 0     TRANS
U86-9       PROBE       5925    1 0     TRANS
U86-9       I/O MODULE  5925    1 0     TRANS
U86-12      PROBE       5FAA    1 0     TRANS
U86-12      I/O MODULE  5FAA    1 0     TRANS
U86-15      PROBE       D909    1 0     TRANS
U86-15      I/O MODULE  D909    1 0     TRANS
U86-16      PROBE       C26B    1 0     TRANS
U86-16      I/O MODULE  C26B    1 0     TRANS
U86-19      PROBE       A814    1 0     TRANS
U86-19      I/O MODULE  A814    1 0     TRANS
```

Figure 4-78: Response File *(video_scan) - continued*

```
program video_init

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! INITIALIZATION PROGRAM for the 2674 Advanced Video Display Controller.!
! The program executes two Master Reset commands followed by the init-  !
! ialization of 15 contiguous Initialization Registers.  Next 6 regis-  !
! ers are initialized which determine the screen memory mapping and the !
! cursor location.                                                      !
!                                                                       !
! This program must be executed before any video testing is performed,  !
! and must be re-executed whenever UUT power has been interrupted.      !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!     (none)                                                            !
!                                                                       !
! GRAPHICS PROGRAMS CALLED:                                             !
!     (none)                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    setspace space (getspace space "i/o", size "byte")

    write ADDR 2, DATA   0     ! Master Reset Command
    write ADDR 2, DATA   0     ! Master Reset Command
    write ADDR 0, DATA $48     ! Write Initialization Register 0
    write ADDR 0, DATA $20     ! Write Initialization Register 1
    write ADDR 0, DATA $22     ! Write Initialization Register 2
    write ADDR 0, DATA $86     ! Write Initialization Register 3
    write ADDR 0, DATA $17     ! Write Initialization Register 4
    write ADDR 0, DATA $4F     ! Write Initialization Register 5
    write ADDR 0, DATA   9     ! Write Initialization Register 6
    write ADDR 0, DATA $28     ! Write Initialization Register 7
    write ADDR 0, DATA   0     ! Write Initialization Register 8
    write ADDR 0, DATA $10     ! Write Initialization Register 9
    write ADDR 0, DATA   0     ! Write Initialization Register 10
    write ADDR 0, DATA   0     ! Write Initialization Register 11
    write ADDR 0, DATA   0     ! Write Initialization Register 12
    write ADDR 0, DATA   0     ! Write Initialization Register 13
    write ADDR 0, DATA   0     ! Write Initialization Register 14
    write ADDR 4, DATA   1     ! Screen Start 1 Lower Register
    write ADDR 6, DATA   0     ! Screen Start 1 Upper Register
    write ADDR 8, DATA   0     ! Cursor Address Lower Register
    write ADDR $A, DATA  0     ! Cursor Address Upper Register
    write ADDR $C, DATA  0     ! Screen Start 2 Lower Register
    write ADDR $E, DATA  0     ! Screen Start 2 Upper Register
    write ADDR 2, DATA $29     ! Enable Screen On Command

end program
```

Figure 4-79: Initialization Program *(video_init)*

```
program video_fill

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! INITIALIZATION PROGRAM fills video RAM with every attribute & char    !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    (none)                                                             !
!                                                                       !
! GRAPHICS PROGRAMS CALLED:                                             !
!    (none)                                                             !
!                                                                       !
! Text Files Accessed:                                                  !
!    vid_fill1                                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   setspace space (getspace space "memory", size "word")
   writeblock file "vid_fill1", format "motorola"

end program
```

Figure 4-80: Initialization Program *(video_fil1)*

```
program video_fil2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! INITIALIZATION PROGRAM to fill video RAM with Non-Blinking characters !
!                                                                      !
! TEST PROGRAMS CALLED:                                                !
!                                                                      !
! GRAPHICS PROGRAMS CALLED:                                            !
!    (none)                                                            !
!                                                                      !
! Text Files Accessed:                                                 !
!    vid_fill2                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   setspace space (getspace space "memory", size "word")
   writeblock file "vid_fill2", format "motorola"

end program
```

Figure 4-81: Initialization Program *(video_fil2)*

Summary of Complete Solution for Video Output 4.7.7.

The entire set of programs and files needed to test and GFI troubleshoot the Video Output functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Video Output)

Programs (PROGRAM):

TEST_VIDEO	Functional Test	Section 4.7.5
VIDEO_FREQ	Stimulus Program	Figure 4-73
VIDEO_OUT	Stimulus Program	Figure 4-75
VIDEO_SCAN	Stimulus Program	Figure 4-77
LEVELS	Stimulus Program	Figure 4-92
VIDEO_INIT	Initialization Program	Figure 4-79
VIDEO_FIL1	Initialization Program	Figure 4-80
VIDEO_FIL2	Initialization Program	Figure 4-81

Stimulus Program Responses (RESPONSE):

VIDEO_FREQ	Figure 4-74
VIDEO_OUT	Figure 4-76
VIDEO_SCAN	Figure 4-78
LEVELS	Figure 4-93

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

VID_FILL1	Initialization Data File
VID_FILL2	Initialization Data File

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

VIDEO CONTROL FUNCTIONAL BLOCK 4.8.

Introduction to Video Control Circuits 4.8.1.

After initialization by the microprocessor, the video control block typically generates four major timing functions:

- Character timing for serializing character or dot graphics information to the Video Output functional block.
- Address generation and timing control for accessing the video RAM.
- Cursor timing and control to the Video Output block.
- Vertical and horizontal sync signals.

The frequency of these signals may vary from about 60 Hz for vertical sync to well over 10 MHz for pixel information. Figure 4-82 shows the timing of some of these signals.

Timing Signals

The vertical scan rate is the measure of how often the entire video picture is drawn on the screen (usually 50 or 60 Hz). The screen is scanned horizontally many times during each vertical scan. If the video display is character-oriented, there might be 10 horizontal scans for each row of characters.

When set up properly, the timing outputs and video RAM address outputs will repeat regularly at the vertical scan rate. All the timing signals (such as the character clock, horizontal scan, blanking, vertical sync, blink rate, and cursor signal) are normally derived from the dot clock.

The cursor timing output is a strobe which occurs when the cursor address is sent out.

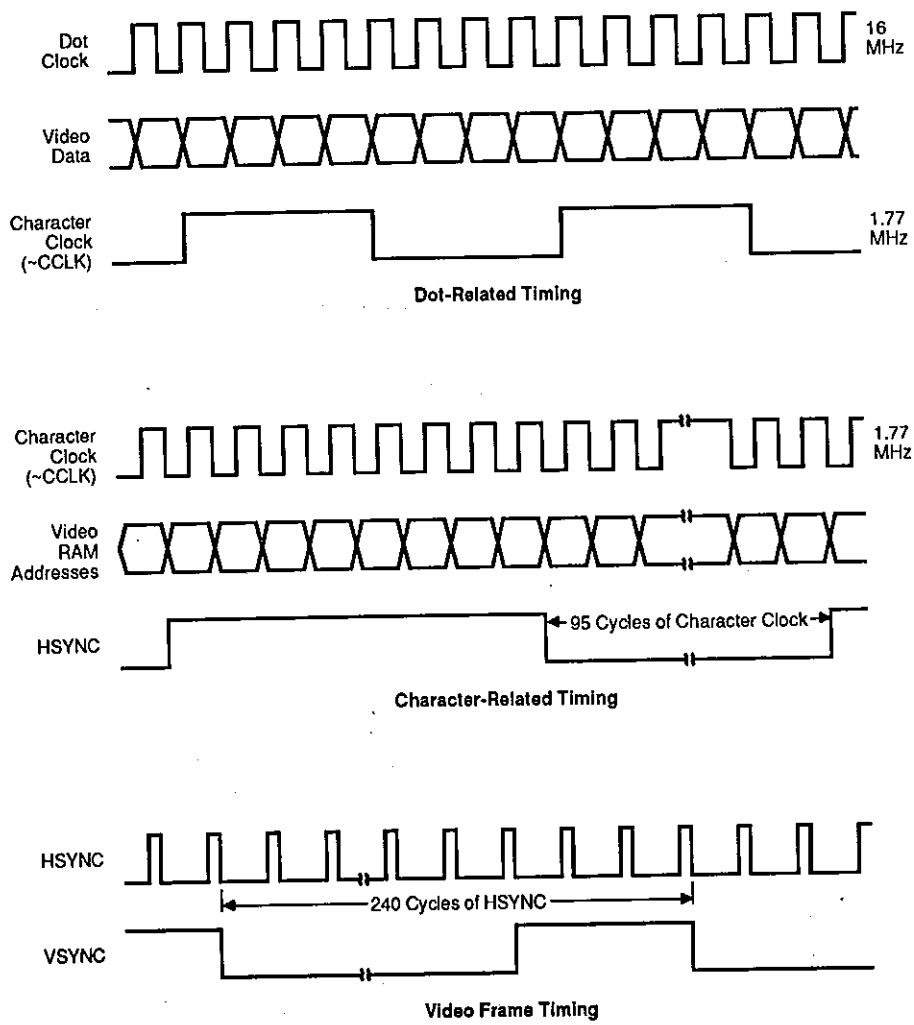


Figure 4-82: Video Display Controller Timing

Considerations for Testing and Troubleshooting 4.8.2.

Video control circuitry can usually be tested in four steps.

1. Initialize the circuitry (set up the video display controller registers if the implementation uses such a chip).

2. Test for proper signature on the scan address lines going to the video RAM to ensure that it cycles through the proper addresses when displaying a frame.

3. Check the vertical and horizontal sync frequency.

   If the timing logic is used in several modes, the three steps described above can be repeated for each mode.

4. Test the cursor strobe generator by clocking from the character clock, starting at the beginning of the frame and stopping at the end of the frame. You may need to test for proper signatures at several cursor positions. For this test approach to work, the cursor cannot be in a blinking mode.

The video RAM access logic, which allows the microprocessor and the video display controller to share video RAM, must arbitrate access to video RAM.

Since the microprocessor and the video display controller are not always synchronous, it may be impossible to find a single clock that gives stable signatures for all of the arbitration logic. One approach to testing the arbitration logic is to count pulses on the outputs of the video control logic while doing a series of writes to video RAM.

The Demo/Trainer UUT contains an example of a memory arbitration circuit which is hard to troubleshoot. It is a state machine with seven inputs and three outputs. In testing this type of circuit, you don't need to worry about how it works. All that

is required is to exercise the inputs in a way that causes a stable response on each output. When this type of circuit does not function, it may be necessary to break some of the feedback loops to isolate the problem to one component. This can be done by using an I/O module to overdrive nodes in the feedback loops.

The character clock will probably be the best clock signal for most of the nodes, including scan address lines, video RAM, and circuitry up to the shift register which converts character information to pixel information. The response measurement should start at the end of the vertical retrace and should stop at the beginning of the vertical retrace. This means that the Start and Stop external control lines from the 9100A/9105A Clock Module or an I/O Module should connect to the vertical sync signal.

Video Control Circuit Example 4.8.3.

The Video Control Circuit of the Demo/Trainer UUT, Figure 4-83, uses a Signetics™ 2674 advanced video display controller (AVDC), U72, for video control. The 2674 is a programmable device designed for use with CRT terminals and display systems that employ raster-scan techniques. It is programmed with CRT-terminal setup information, providing cursor, blanking, and clock signals to the 2675 Attributes Controller chip (U78) in the Video Output functional block.

The 2674 outputs to the Video RAM functional block on the scan address lines DADD00-11 in synchronization with the horizontal and vertical sync signals.

The remaining circuitry in this block is a state machine. It is normally inactive, but upon writing to video RAM it produces a variable-length wait state to synchronize the microprocessor bus cycle to the video character clock.

Figure 4-83 shows a timing diagram for the video control circuit of the Demo/Trainer UUT.

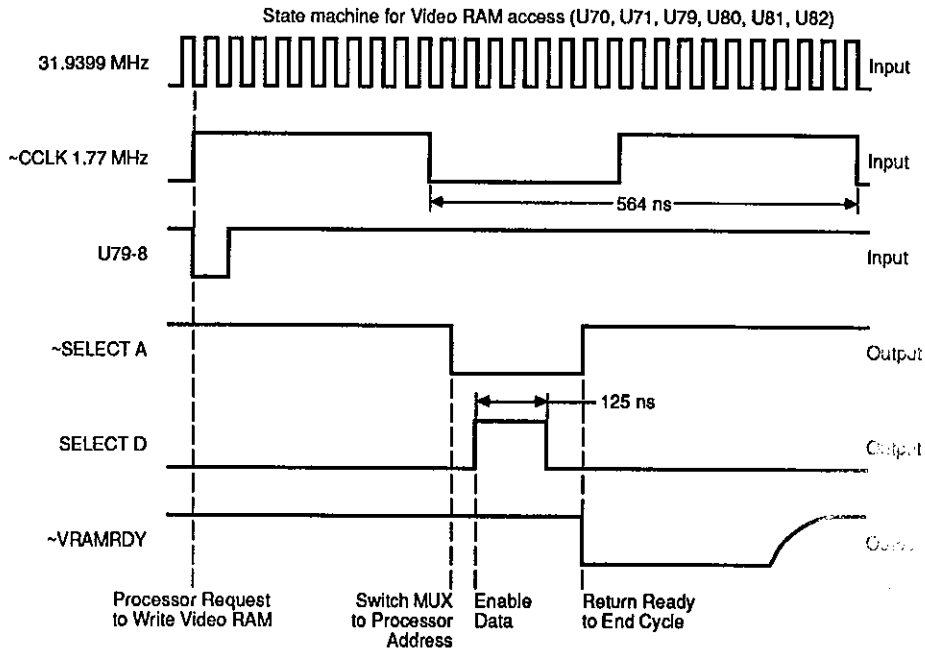


Figure 4-83: Video Control Functional Block Timing

Keystroke Functional Test 4.8.4.

Part A:

1. Clip a 40-pin clip module on I/O module 1 to test U72.

2. Use the the EXEC, I/O MOD, and SOFT KEYS keys with the following commands and check the measured frequency with the correct frequency ranges shown in the response table of Figure 4-84.

```
EXECUTE UUT DEMO PROGRAM VIDEO_INIT
FREQ ON I/O MOD 1 PIN <see response table>
```

Part B:

1. Connect the external control lines of the I/O module 1 as follows:

   Clock to CCLK (U78-33)
   Start to VSYNC (U88-13)
   Stop to VSYNC (U88-13)
   Enable to BLANK (U78-12)

2. Use the EXEC, SYNC, and I/O MOD keys with the following commands, and check the measurements with the response table in Figure 4-85.

```
EXECUTE UUT DEMO PROGRAM VIDEO_INIT
SYNC I/O MOD 1 TO EXT ENABLE LOW ...
... CLOCK ↓START ↓ STOP ↑
ARM  I/O MOD 1 FOR CAPTURE USING SYNC
SHOW I/O MOD 1 PIN <see response table> ...
... CAPTURED RESPONSES
```

NOTE

The SHOW command requires a clip module pin number rather than a part pin number. This requires you to translate part pin numbers to clip module pin numbers (see Appendix B of the Technical User's Manual). For your convenience, this translation has been done for you in this example, and the results are shown in the "I/O MOD PIN" column of the response table in the Figure 4-85.

Part C:

Use the SYNC, PROBE, and WRITE keys with the probe to test the video ready signals. Compare the results with the response table in Figure 4-86.

```
SYNC PROBE TO POD DATA
ARM PROBE FOR CAPTURE USING SYNC
WRITE BLOCK INTO MEMORY FROM UUT DEMO ...
... FILE VID_FILL1 USING MOTOROLA
... ,(ADDR OPTION: MEMORY WORD)
SHOW PROBE CAPTURED RESPONSES
```

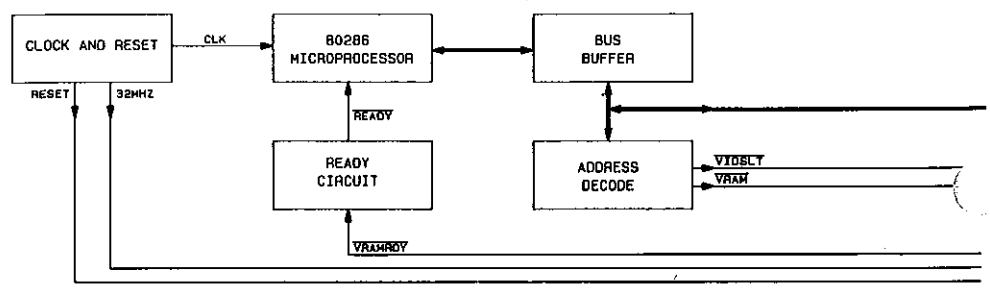
Keystroke Functional Test (Part A)

CONNECTION TABLE

	MEASUREMENT
(NONE)	I/O MOD U72

RESPONSE TABLE

SIGNAL	PIN	I/O MOD PIN	FREQUENCY MINIMUM	FREQUENCY MAXIMUM
CCLK	U72-16	16	1.775 MHZ	1.780 MHZ
BLANK	-17	17	14426 HZ	14435 HZ
VSYNC	-18	18	58 HZ	62 HZ
HSYNC	-19	19	16766 HZ	16774 HZ



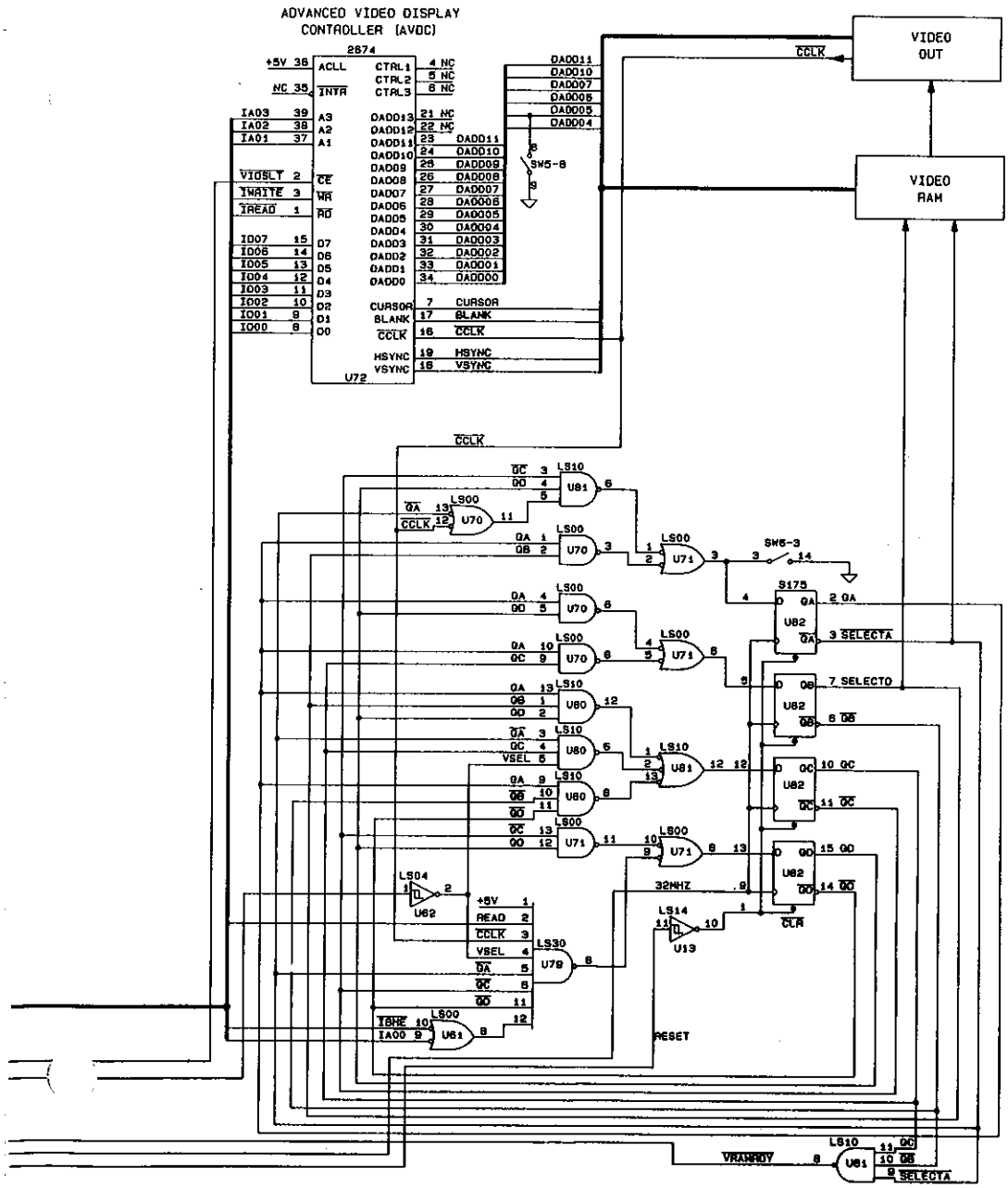


Figure 4-84: Video Control Functional Test (Part A)

Keystroke Functional Test (Part B)

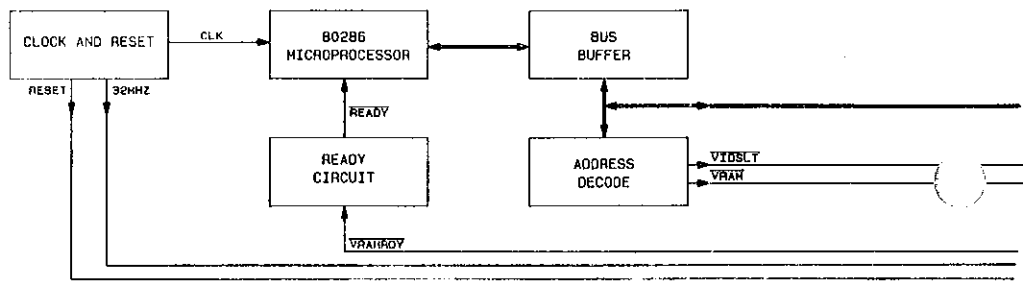
CONNECTION TABLE

STIMULUS	MEASUREMENT CONTROL	MEASUREMENT
(NONE)	I/O MOD	I/O MOD
	CLOCK U78-33 START U88-13 STOP U88-13 ENABLE U78-12	U72

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
DAD00	U72-34	34	4155
DAD01	-33	33	3F33
DAD02	-32	32	A65A
DAD03	-31	31	9024
DAD04	-30	30	DE6D
DAD05	-29	29	D6FA
DAD06	-28	28	7AC3
DAD07	-27	27	0477
DAD08	-26	26	E941
DAD09	-25	25	88B8
DAD10	-24	24	60B0
DAD11	-23	23	D869 or 0000*

*DAD11 has a signature of D869 one half of the time and 0000 the other half of the time.



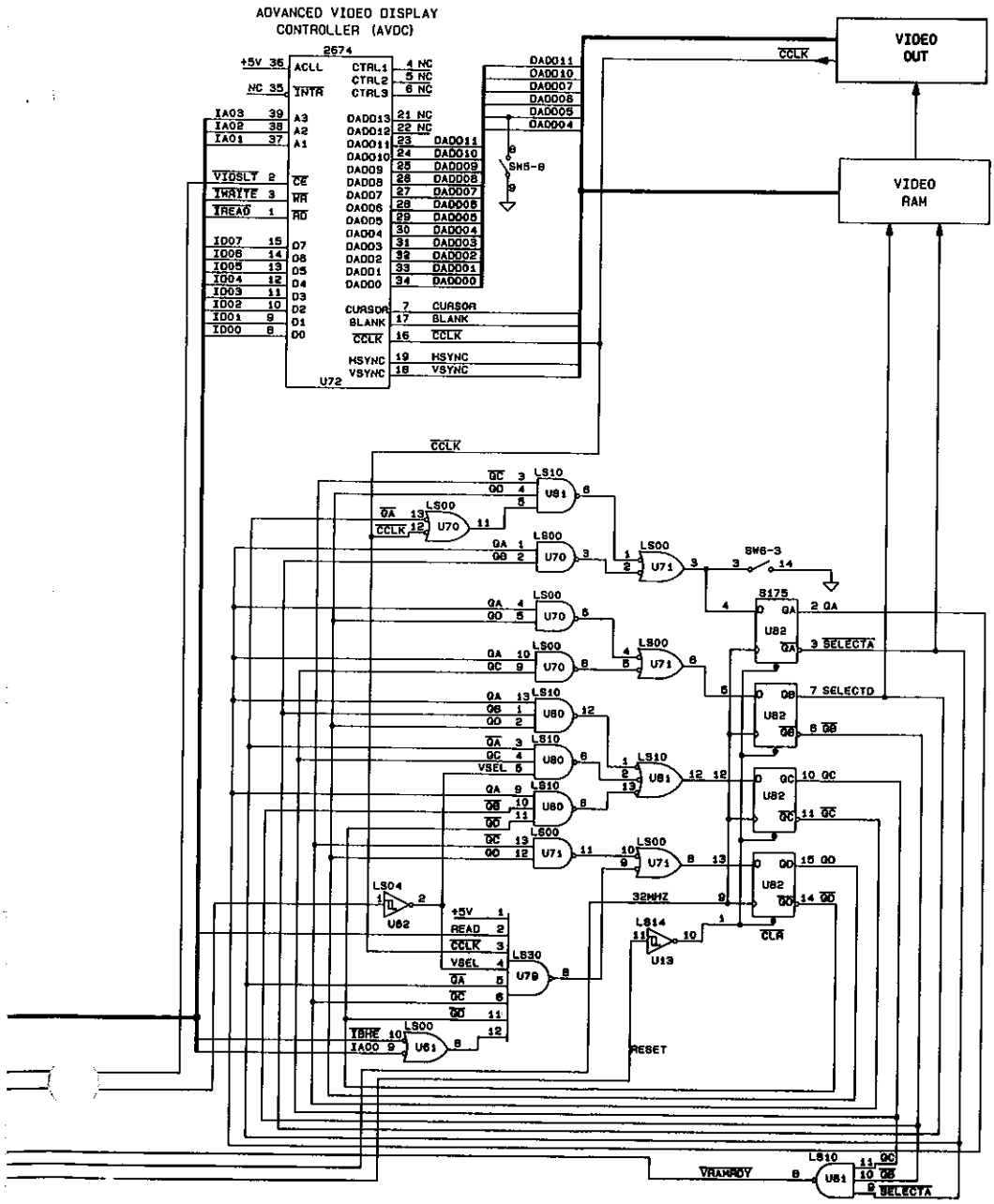


Figure 4-85: Video Control Functional Test (Part B)

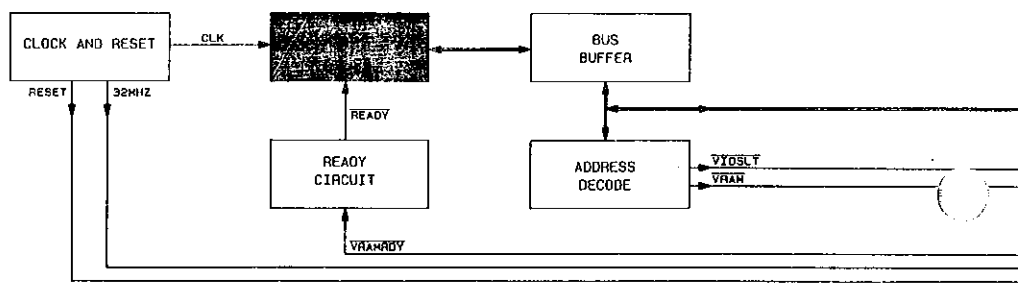
Keystroke Functional Test (Part C)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted]	PROBE U82-3 U81-8

RESPONSE TABLE

SIGNAL	PART PIN	SIGNATURE	TRANS COUNT
SELECTA VRAMRDY	U82-3 U81-8	7A70 0000	2048 2048



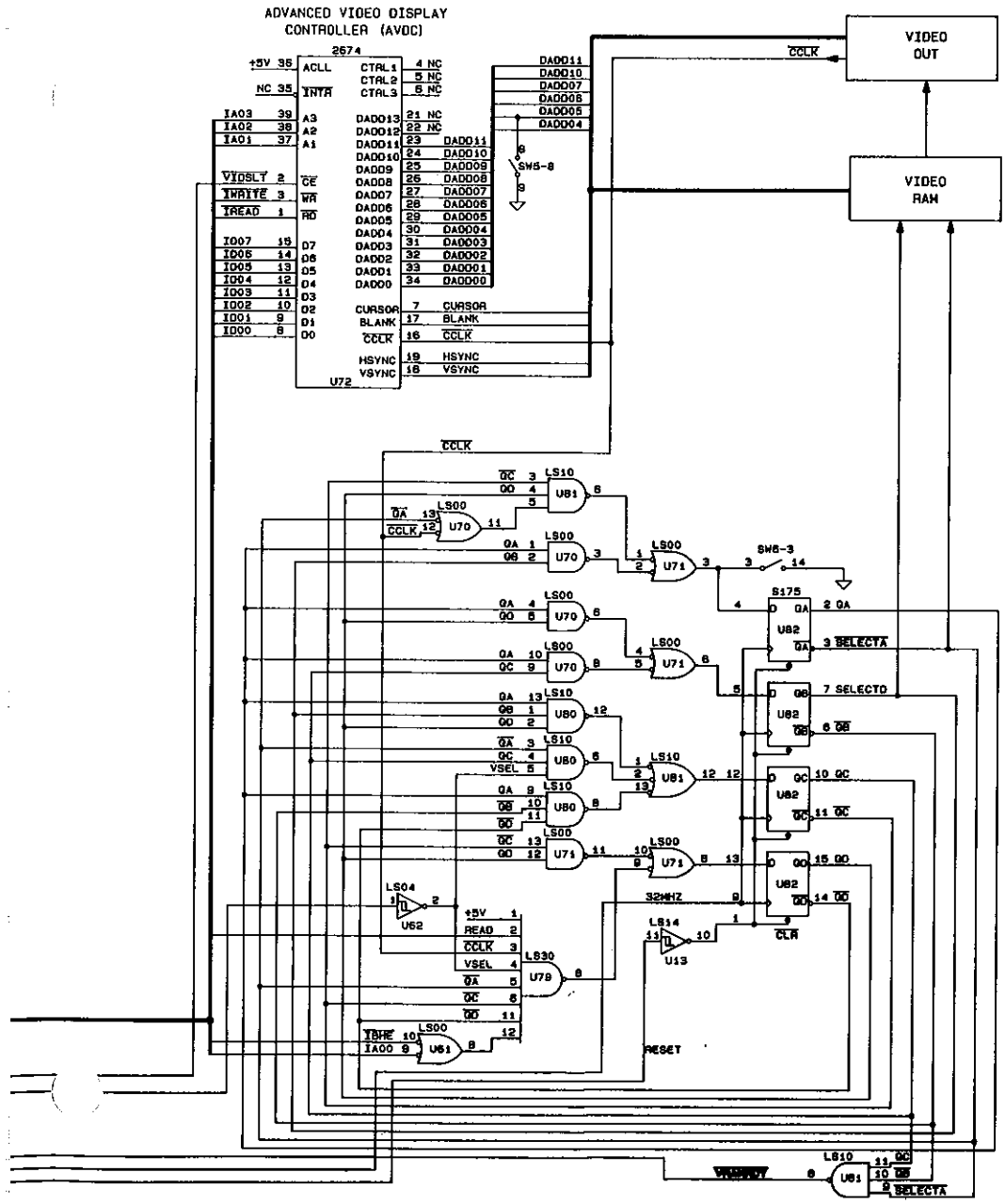


Figure 4-86: Video Control Functional Test (Part C)

Programmed Functional Test 4.8.5.

The *tst_vidctl* program is the programmed functional test for the Video Control functional block. This program checks the video controller IC (U72) and the video RAM ready generator outputs U81-8 and U82-3 using the *gfi test* command. If the *gfi test* command fails, the *abort_test* program is executed and GFI troubleshooting begins. (See the Bus Buffer functional block for a discussion of the *abort_test* program).

```
program tst_vidctl

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the VIDEO CONTROL functional block.                !
!                                                                       !
! This program tests the VIDEO CONTROL functional block of the          !
! Demo/Trainer.  The gfi test command and I/O module are used to        !
! perform the test.                                                     !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    abort_test (ref-pin)                   If gfi has an accusation    !
!                                           display the accusation else !
!                                           create a gfi hint for the   !
!                                           ref-pin and terminate the test!
!                                           program (GFI begins trouble- !
!                                           shooting).                  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Setup

   print "\nl\nlTESTING VIDEO CONTROL Circuit"

! Main part of test

   podsetup 'enable ~ready' "on"

   if gfi test "U72-34" fails then abort_test("U72-34")
   if gfi test "U81-8" fails then abort_test("U81-8")
   if gfi test "U82-3" fails then abort_test("U82-3")

   print "VIDEO CONTROL TEST PASSES"
end program
```

Stimulus Programs and Responses 4.8.6.

Figure 4-87 is the stimulus program planning diagram for the Video Control functional block. The *video_data* stimulus program outputs data onto the data bus. The *video_freq* stimulus program initializes the video registers and then measures frequency. The *video_scan* stimulus program initializes video RAM by executing *video_fill*, which fills video

RAM with characters including blinking characters. The *reset_low* stimulus program prompts the test operator to push the Demo/Trainer UUT RESET pushbutton and measures the level of the reset signal. The *levels* stimulus program stimulates activity appropriate for measuring static levels on a number of nodes in the Video RAM Ready (VRAMRDY) generation circuit. The *video_rdy* stimulus program stimulates the Video RAM Ready (VRAMRDY) generation circuit by writes made to the write-only video RAM.

All the stimulus programs execute *video_init* before any measurements are made on the video circuitry.

Stimulus Program Planning

PROGRAM: RESET_LOW
PROMPTS THE OPERATOR TO PRESS THE RESET KEY AND THEN CHECKS FOR A LOW LEVEL
MEASUREMENT AT: U13-10

[redacted]

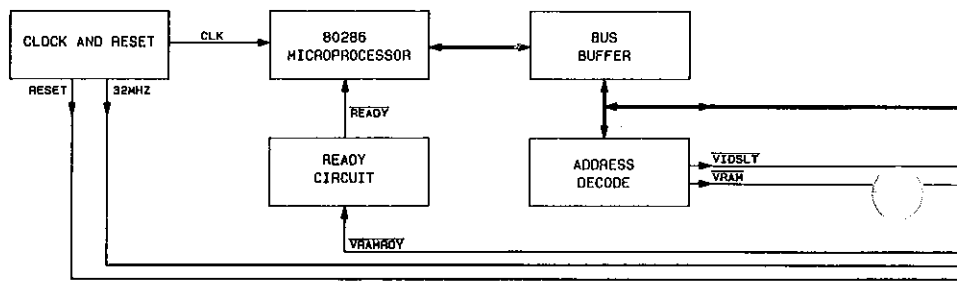
PROGRAM: VIDEO_FREQ
EXECUTES VIDEO_INIT AND MEASURES FREQUENCY
MEASUREMENT AT: U72-17,19,18 U70-11

[redacted]

PROGRAM: VIDEO_SCAN
EXECUTES VIDEO_INIT, VIDEO_FIL1, AND MEASURES ALL CIRCUITRY WHERE DATA IS CLOCKED THROUGH BY CHARACTERS
MEASUREMENT AT: U72-24,25,32,31,30,29,28,27,26,25,24,23,7

[redacted]

INITIALIZATION PROGRAM: VIDEO_INIT
INITIALIZES VIDEO REGISTERS TO STANDARD OPERATING MODE
MEASUREMENT AT: (NONE)



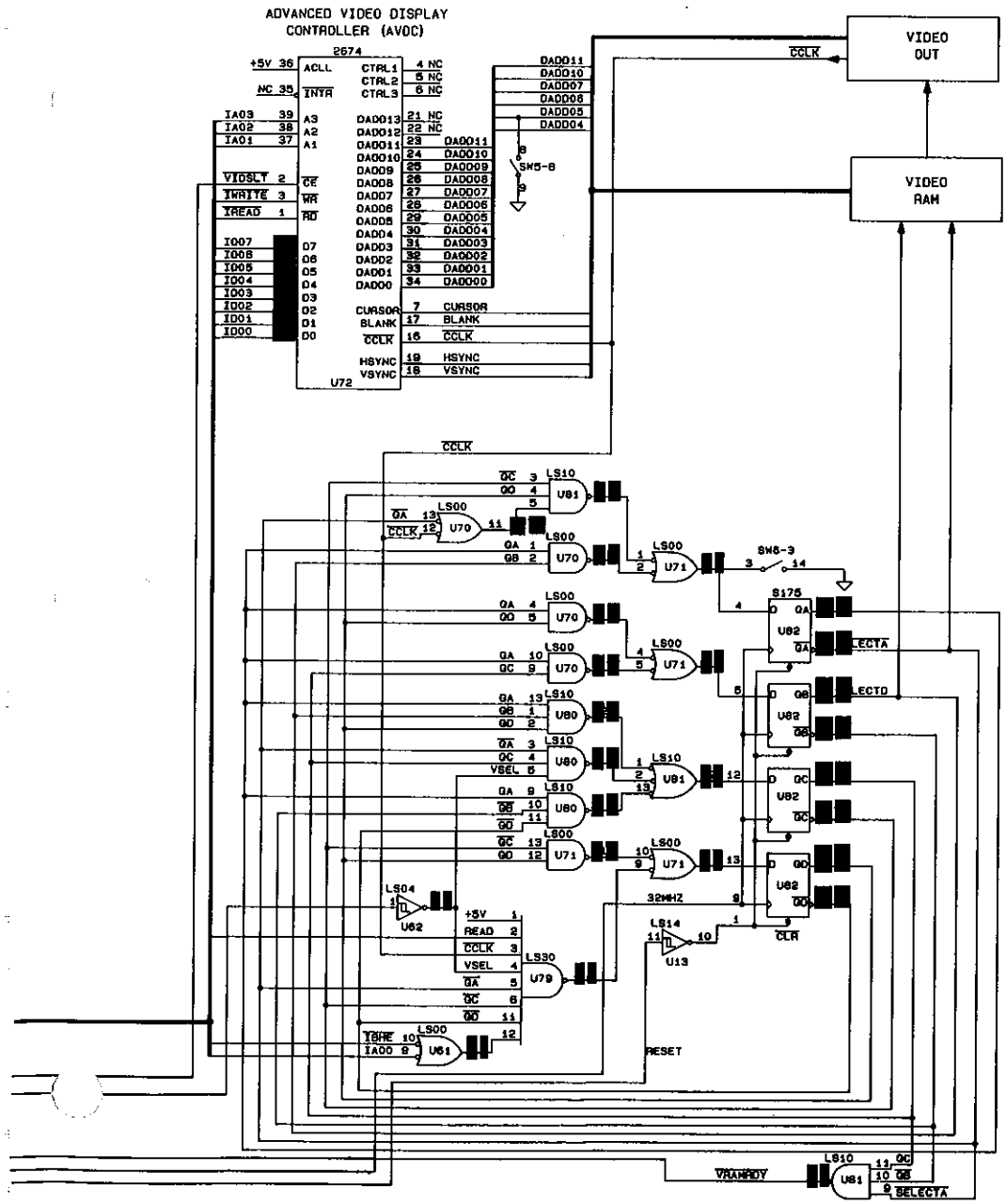


Figure 4-87: Video Control Stimulus Program Planning

```
program video_data

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to extract data from U72 registers.                    !
!                                                                         !
! Stimulus programs and response files are used by GFI to backtrace      !
! from a failing node.  The stimulus program must create repeatable UUT  !
! activity and the response file contains the known-good responses for   !
! the outputs in the UUT that are stimulated by the stimulus program.    !
!                                                                         !
! This stimulus program is one of the programs which creates activity    !
! in the kernel area of the UUT.  These programs create activity with    !
! or without the ready circuit working properly.  Because of this, all   !
! the stimulus programs in the kernel area must disable the READY input  !
! to the pod, then perform the stimulus, then re-enable the READY input  !
! to the pod.  The 80286 microprocessor has a separate bus controller;   !
! for this reason, disabling ready and performing stimulus can get the   !
! bus controller out of synchronization with the pod.  Two fault         !
! handlers trap pod timeout conditions that indicate the bus controller  !
! is out of synchronization.  The recover() program is executed to       !
! resynchronize the bus controller and the pod.                          !
!                                                                         !
! TEST PROGRAMS CALLED:                                                   !
!    recover     ()                         The 80286 microprocessor has a!
!                                           bus controller that is totaly !
!                                           separate from the pod.  In    !
!                                           some cases the pod can get out!
!                                           of sync with the bus control- !
!                                           ler.  The recover program     !
!                                           resynchronizes the pod and the!
!                                           bus controller.               !
!                                                                         !
! GRAPHICS PROGRAMS CALLED:                                               !
!    (none)                                                               !
!                                                                         !
! Local Variables Modified:                                               !
!    recover_times                          Reset to Zero                 !
!    devname                                Measurement device            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle
```

(continued on the next page)

Figure 4-88: Stimulus Program *(video_data)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine measurement device.

    if (gfi control) = "yes" then
        devname = gfi device
    else
        devname = "/mod1"
    end if
   print "Stimulus Program VIDEO_DATA"

! Set addressing mode and initialize.

    option = getspace type "i/o", size "byte"
    setspace( option )
    write ADDR 8, DATA $FF      ! Cursor Address Lower
    write ADDR $A, DATA  0      ! Cursor Address Upper
    write ADDR $C, DATA $AA     ! Screen Start 2 Lower
    write ADDR $E, DATA $35     ! Screen Start 2 Upper

! Setup measurement device.

    reset device devname
    sync device devname, mode "pod"
    sync device "/pod", mode "data"

! Present stimulus to UUT.

    arm device devname          ! Start response capture
       read addr  8             !    Lower Cursor Addr Reg
       read addr  $A            !    Upper Cursor Addr Reg
       read addr  $C            !    Lower Screen Start 2
       read addr  $E            !    Upper Screen Start 2
    readout device devname      ! End response capture

end program
```

Figure 4-88: Stimulus Program *(video_data) - continued*

STIMULUS PROGRAM NAME: VIDEO_DATA
DESCRIPTION: SIZE: 318 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U72-8	PROBE	0009		1 0	TRANS		
U72-8	I/O MODULE	0009		1 0	TRANS		
U72-9	PROBE	000A		1 0	TRANS		
U72-9	I/O MODULE	000A		1 0	TRANS		
U72-10	PROBE	0009		1 0	TRANS		
U72-10	I/O MODULE	0009		1 0	TRANS		
U72-11	PROBE	000A		1 0	TRANS		
U72-11	I/O MODULE	000A		1 0	TRANS		
U72-12	PROBE	0009		1 0	TRANS		
U72-12	I/O MODULE	0009		1 0	TRANS		
U72-13	PROBE	000B		1 0	TRANS		
U72-13	I/O MODULE	000B		1 0	TRANS		
U72-14	PROBE	0008		1 0	TRANS		
U72-14	I/O MODULE	0008		1 0	TRANS		
U72-15	PROBE	000A		1 0	TRANS		
U72-15	I/O MODULE	000A		1 0	TRANS		

Figure 4-89: Response File *(video_data)*

```
program video_rdy

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM activates video ready circuitry.                     !
!                                                                       !
! Stimulus programs and response files are used by GFI to backtrace     !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!     (none)                                                            !
!                                                                       !
! GRAPHICS PROGRAMS CALLED:                                             !
!     (none)                                                            !
!                                                                       !
! Local Variables Modified:                                             !
!    devname                                  Measurement device        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine measurement device.

    if (gfi control) = "yes" then
        devname = gfi device
    else
        devname = "/probe"
    end if
    print "Stimulus Program VIDEO_RDY"

! Set addressing mode and Set up measurement device.

    reset device devname
    setspace space (getspace space "memory", size "word")
    sync device devname, mode "pod"
    sync device "/pod", mode "data"

! Present stimulus to UUT

    arm device devname                          ! Start response capture.
      toggledata addr $20000, data 0, mask $FFFF ! Create a burst of writes.
    readout device devname                      ! End response capture.

end program
```

Figure 4-90: Stimulus Program *(video_rdy)*

STIMULUS PROGRAM NAME: VIDEO_RDY
DESCRIPTION: SIZE: 1,411 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Response Data: Clk LVL	Response Data: Counter Mode	Counter Range	Priority Pin
U82-2	PROBE	0000	1 0		TRANS		
U82-2	I/O MODULE	0000	1 0		TRANS		
U82-3	PROBE	3951	1 0		TRANS		
U82-3	I/O MODULE	3951	1 0		TRANS		
U82-7	PROBE	0000	1 0		TRANS		
U82-7	I/O MODULE	0000	1 0		TRANS		
U82-6	PROBE	3951	1 0		TRANS		
U82-10	PROBE	3951	1 0		TRANS		
U82-10	I/O MODULE	3951	1 0		TRANS		
U82-11	PROBE	0000	1 0		TRANS		
U82-11	I/O MODULE	0000	1 0		TRANS		
U82-15	PROBE	0000	1 0		TRANS		
U82-15	I/O MODULE	0000	1 0		TRANS		
U82-14	PROBE	3951	1 0		TRANS		
U82-14	I/O MODULE	3951	1 0		TRANS		
U81-6	PROBE	3951	1 0		TRANS		
U81-6	I/O MODULE	3951	1 0		TRANS		
U81-8	PROBE	0000	1 0		TRANS		
U81-8	I/O MODULE	0000	1 0		TRANS		
U81-12	PROBE	3951	1 0		TRANS		
U80-6	PROBE	0000	1 0		TRANS		
U80-8	PROBE	3951	1 0		TRANS		
U80-12	PROBE	3951	1 0		TRANS		
U79-8	I/O MODULE	3951	1 0	1	TRANS		
U71-3	PROBE	0000	1 0		TRANS		
U71-3	I/O MODULE	0000	1 0		TRANS		
U71-6	PROBE	0000	1 0		TRANS		
U71-6	I/O MODULE	0000	1 0		TRANS		
U71-8	PROBE	0000	1 0		TRANS		
U71-8	I/O MODULE	0000	1 0		TRANS		
U71-11	I/O MODULE	3951	1 0		TRANS		
U70-3	I/O MODULE	3951	1 0		TRANS		
U70-6	I/O MODULE	3951	1 0		TRANS		
U70-8	I/O MODULE	3951	1 0		TRANS		
U70-11	PROBE		1 0		TRANS		
U70-11	I/O MODULE		1 0		TRANS		
U62-2	PROBE	3951	1 0		TRANS		
U62-2	I/O MODULE	3951	1 0		TRANS		
U62-6	I/O MODULE	0000	1 0		TRANS		
U62-10	I/O MODULE	3951	1		TRANS		
U62-12	I/O MODULE	3951	1		TRANS		
U61-6	I/O MODULE	3951	1 0		TRANS		
U61-3	I/O MODULE	3951	1 0		TRANS		
U61-8	I/O MODULE	3951	1		TRANS		

(continued on the next page)

Figure 4-91: Response File *(video_rdy)*

U84-4	I/O MODULE	1 0	TRANS	8300-9500
U84-7	I/O MODULE	1 0	TRANS	14000-17500
U84-9	I/O MODULE	1 0	TRANS	30000-36000
U84-12	I/O MODULE	1 0	TRANS	61000-71000
U83-4	I/O MODULE	1 0	TRANS	950-1300
U83-7	I/O MODULE	1 0	TRANS	1400-1800
U83-9	I/O MODULE	1 0	TRANS	2300-2700
U83-12	I/O MODULE	1 0	TRANS	4100-4700
U73-7	I/O MODULE	1 0	TRANS	475-800
U73-9	I/O MODULE	1 0	TRANS	500-900
U73-12	I/O MODULE	1 0	TRANS	700-1000
U69-18	I/O MODULE	1 0	TRANS	
U69-16	I/O MODULE	1 0	TRANS	
U69-14	I/O MODULE	1 0	TRANS	
U69-12	I/O MODULE	1 0	TRANS	
U69-9	I/O MODULE	1 0	TRANS	
U69-7	I/O MODULE	1 0	TRANS	
U69-5	I/O MODULE	1 0	TRANS	
U69-3	I/O MODULE	1 0	TRANS	
U68-18	I/O MODULE	1 0	TRANS	
U68-16	I/O MODULE	1 0	TRANS	
U68-14	I/O MODULE	1 0	TRANS	
U68-12	I/O MODULE	1 0	TRANS	
U68-9	I/O MODULE	1 0	TRANS	
U68-7	I/O MODULE	1 0	TRANS	
U68-5	I/O MODULE	1 0	TRANS	
U68-3	I/O MODULE	1 0	TRANS	

Figure 4-91: Response File *(video_rdy) - continued*

```
program levels

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to measure level history.                                !
!                                                                           !
! Stimulus programs and response files are used by GFI to backtrace        !
! from a failing node.  The stimulus program must create repeatable UUT    !
! activity and the response file contains the known-good responses for     !
! the outputs in the UUT that are stimulated by the stimulus program.      !
!                                                                           !
! This is a general purpose routine that is used to characterize the       !
! level history both sync and async of a node that may not lend itself     !
! to signatures or frequency.                                               !
!                                                                           !
! TEST PROGRAMS CALLED:                                                     !
!    (none)                                                                 !
!                                                                           !
! GRAPHICS PROGRAMS CALLED:                                                 !
!    (none)                                                                 !
!                                                                           !
! Local Variables Modified:                                                 !
!    devname                                   Measurement device           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_no_clk
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI user select which I/O module to use.

   if (gfi control) = "yes" then
     devname = gfi device
   else
     devname = "/mod1"
   end if
   print "Stimulus Program LEVELS"

! Set desired measurement modes.

   reset device devname

! No stimulus is applied; response is async levels.

   arm device devname        ! Start response capture.
   readout device devname    ! End response capture

end levels
```

Figure 4-92: Stimulus Program *(levels)*

STIMULUS PROGRAM NAME: LEVELS
DESCRIPTION: SIZE: 1,435 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U82-2	PROBE		0		TRANS		
U82-2	I/O MODULE		0		TRANS		
U82-3	PROBE		1		TRANS		
U82-3	I/O MODULE		1		TRANS		
U82-7	PROBE		0		TRANS		
U82-7	I/O MODULE		0		TRANS		
U82-6	PROBE		1		TRANS		
U82-10	PROBE		0		TRANS		
U82-10	I/O MODULE		0		TRANS		
U82-11	PROBE		1		TRANS		
U82-11	I/O MODULE		1		TRANS		
U82-15	PROBE		0		TRANS		
U82-15	I/O MODULE		0		TRANS		
U82-14	PROBE		1		TRANS		
U82-14	I/O MODULE		1		TRANS		
U81-6	PROBE		1		TRANS		
U81-6	I/O MODULE		1		TRANS		
U81-8	PROBE		1		TRANS		
U81-8	I/O MODULE		1		TRANS		
U81-12	PROBE		0		TRANS		
U80-6	PROBE		1		TRANS		
U80-8	PROBE		1		TRANS		
U80-12	PROBE		1		TRANS		
U79-8	I/O MODULE		1		TRANS		
U71-3	PROBE		0		TRANS		
U71-3	I/O MODULE		0		TRANS		
U71-6	PROBE		0		TRANS		
U71-6	I/O MODULE		0		TRANS		
U71-8	PROBE		0		TRANS		
U71-8	I/O MODULE		0		TRANS		
U71-11	I/O MODULE		1		TRANS		
U70-3	I/O MODULE		1		TRANS		
U70-6	I/O MODULE		1		TRANS		
U70-8	I/O MODULE		1		TRANS		
U70-11	PROBE		1	0	TRANS		
U70-11	I/O MODULE		1	0	TRANS		
U62-2	PROBE		0		TRANS		
U62-2	I/O MODULE		0		TRANS		
U61-8	I/O MODULE		1		TRANS		
U62-6	I/O MODULE		0		TRANS		
U61-3	I/O MODULE		1		TRANS		
U61-6	I/O MODULE		1		TRANS		
U84-4	I/O MODULE		1	0	TRANS		
U84-7	I/O MODULE		1	0	TRANS		

(continued on the next page)

Figure 4-93: Response File *(levels)*

U84-9	I/O MODULE	1 0	TRANS
U84-12	I/O MODULE	1 0	TRANS
U83-4	I/O MODULE	1 0	TRANS
U83-7	I/O MODULE	1 0	TRANS
U83-9	I/O MODULE	1 0	TRANS
U83-12	I/O MODULE	1 0	TRANS
U73-7	I/O MODULE	1 0	TRANS
U73-9	I/O MODULE	1 0	TRANS
U73-12	I/O MODULE	1 0	TRANS
U69-18	I/O MODULE	1 0	TRANS
U69-16	I/O MODULE	1 0	TRANS
U69-14	I/O MODULE	1 0	TRANS
U69-12	I/O MODULE	1 0	TRANS
U69-9	I/O MODULE	1 0	TRANS
U69-7	I/O MODULE	1 0	TRANS
U69-5	I/O MODULE	1 0	TRANS
U69-3	I/O MODULE	1 0	TRANS
U68-18	I/O MODULE	1 0	TRANS
U68-16	I/O MODULE	1 0	TRANS
U68-14	I/O MODULE	1 0	TRANS
U68-12	I/O MODULE	1 0	TRANS
U68-9	I/O MODULE	1 0	TRANS
U68-7	I/O MODULE	1 0	TRANS
U68-5	I/O MODULE	1 0	TRANS
U68-3	I/O MODULE	1 0	TRANS
J4-6	PROBE	0	TRANS
J4-6	I/O MODULE	0	TRANS
J4-10	PROBE	1	TRANS
J4-10	I/O MODULE	1	TRANS
R34-1	PROBE	1	TRANS
DS1-2	PROBE	1	TRANS
R26-1	PROBE	0	TRANS
R26-1	I/O MODULE	0	TRANS
R32-1	PROBE	1	TRANS
R4-1	PROBE	0	TRANS
R61-1	PROBE	X	TRANS
R77-1	PROBE	1	TRANS
R78-2	PROBE	1	TRANS
R79-2	PROBE	1	TRANS
R80-1	PROBE	1	TRANS
U26-3	I/O MODULE	X	TRANS
U13-4	PROBE	1	TRANS
U13-4	I/O MODULE	1	TRANS
U13-12	PROBE	0	TRANS
U13-12	I/O MODULE	0	TRANS
C13-1	PROBE	1X0	TRANS
C4-1	PROBE	0	TRANS
U14-65	PROBE	1	TRANS
U14-65	I/O MODULE	1	TRANS

Figure 4-93: Response File *(levels) - continued*

Summary of Complete Solution for Video Control 4.8.7.

The entire set of programs and files needed to test and GFI troubleshoot the Video Control functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Video Control)

Programs (PROGRAM):

TST_VCTRL	Functional Test	Section 4.8.5
RESET_LOW	Stimulus Program	Figure 4-115
VIDEO_DATA	Stimulus Program	Figure 4-88
VIDEO_FREQ	Stimulus Program	Figure 4-73
VIDEO_RDY	Stimulus Program	Figure 4-90
VIDEO_SCAN	Stimulus Program	Figure 4-77
LEVELS	Stimulus Program	Figure 4-92
VIDEO_INIT	Initialization Program	Figure 4-79

Stimulus Program Responses (RESPONSE):

RESET_LOW	Figure 4-116
VIDEO_DATA	Figure 4-89
VIDEO_FREQ	Figure 4-74
VIDEO_RDY	Figure 4-91
VIDEO_SCAN	Figure 4-78
LEVELS	Figure 4-93

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

(This page is intentionally blank.)

VIDEO RAM FUNCTIONAL BLOCK 4.9.

Introduction to Video RAM 4.9.1.

Video RAM blocks come in several forms. Here are some of the common configurations:

- Character-oriented video RAM, with secondary character-generation ROM or RAM.
- Pixel-oriented video RAM.
- Combinations of the above.

Access to video RAM can be provided in several ways, including:

- The video display controller may directly access microprocessor memory by stealing memory cycles.
- Video RAM may be separate but still mapped into microprocessor memory space. In this case, access to this memory may be write-only or read/write.
- Access to video RAM may be through I/O-mapped registers.
- If character-generation RAM is used, access to character RAM may be different than access to video RAM.

Considerations for Testing and Troubleshooting 4.9.2.

Testing of video display circuits is complicated by the fact that there may be as many as three separate hierarchical memory spaces, each of which may be sectioned for use only in a particular mode of operation:

- Video RAM
- Character ROM or RAM
- Color palette RAM

Video RAM

If video RAM has read/write access and is mapped into the microprocessor memory space, it can be tested with the 9100A/9105A's built-in RAM test (Section 4.4 discusses this built-in test). If video RAM does not have read access, the video RAM output must be tested with the I/O module or the probe. The 9100A/9105A external Start and Stop control lines should be connected (probably to vertical sync) so that one frame is captured. The 9100A/9105A external Clock control lines should be connected to the appropriate clock signal so that valid RAM output will be captured for each read cycle.

With the above connections, the following procedure will usually test video RAM:

1. Initialize the video circuitry, if not already initialized.

2. Initialize the video RAM with blinking enabled. The TL/1 *writeblock* and *writefill* commands can be used to do this.

3. Set the video control mode so that it accesses as much video RAM as possible.

4. Measure signatures at the video RAM output and compare them to good signatures.

5. Steps 2, 3, and 4 can be repeated, varying the test pattern loaded into video RAM. For example, with 16-bit-wide memory try test patterns like FFFF, 0000, 7777, and AAAA, or ramping data over the entire video RAM.

Character ROM or RAM

If the video RAM is character oriented, with secondary character ROM or RAM, a pattern can be written into the video RAM that cycles through the character-memory addresses. In the case of character ROM, signatures collected at the ROM outputs serve to test the ROM. In the case of character RAM, a pattern must be loaded into the RAM before testing.

Video RAM Circuit Example 4.9.3.

Figure 4-94 shows the Video RAM functional block for the Demo/Trainer UUT. Components U74 and U85 provide 2K bytes of static video RAM. When addressed over the main address bus (IA01-11), video RAM is used to store ASCII character codes supplied by the microprocessor over the main data bus (DB00-15). The system is character-mapped: a specific video RAM address maps into a physical location on the monitor screen.

The video control logic sequentially samples these addresses over lines DADD00-11 to generate display characters using the ASCII codes at these addresses and the corresponding display-character information in the character PROM (see U77 in the Video Output functional block).

The multiplexers U73, U83, and U84 select between the video control address lines (DAD00-11) and the buffered microprocessor lines (IA01-11). The selection control for this multiplexing comes from the Video Control functional block.

Keystroke Functional Test 4.9.4.

1. Connect the external control lines of I/O module 1 as follows:

   Clock to CCLK (U78-33)
   Start to VSYNC (U88-13)
   Stop to VSYNC (U88-13)
   Enable to BLANK (U78-12)

2. Use a 24-pin clip module on side A of I/O module 1 to test the video scan signal. Use the EXEC, SYNC, and I/O MOD keys to enter the following commands. Then, compare the measurements with the response tables in Figure 4-94.

```
EXECUTE UUT DEMO PROGRAM VIDEO_INIT
EXECUTE UUT DEMO PROGRAM VIDEO_FIL1
SYNC I/O MOD 1 TO EXT ENABLE LOW ...
... CLOCK ↓ START ↓ STOP ↑
ARM  I/O MOD 1 FOR CAPTURE USING SYNC
SHOW I/O MOD 1 PIN <see response table> ...
... CAPTURED RESPONSES
```

NOTE

The SHOW command requires a clip module pin number rather than a part pin number. This requires you to translate part pin numbers to clip module pin numbers (see Appendix B of the Technical User's Manual). For your convenience, this translation has been done for you in this example, and the results are shown in the "I/O MOD PIN" column of the response table in Figure 4-94.

(This page is intentionally blank.)

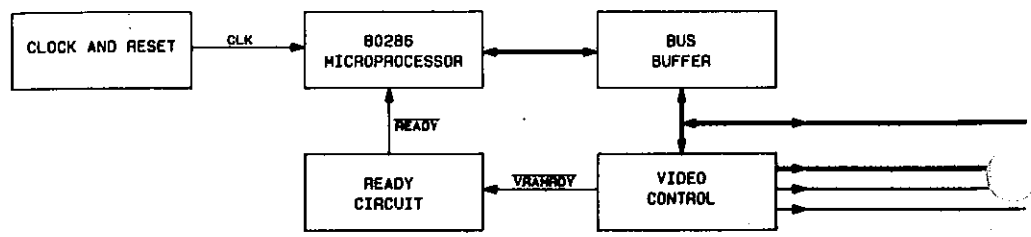
Keystroke Functional Test

CONNECTION TABLE

	MEASUREMENT CONTROL		MEASUREMENT
(NONE)	I/O MOD		I/O MOD
	CLOCK	U78-33	U74
	START	U88-13	U85
	STOP	U88-13	
	ENABLE	U78-12	

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
DB00	U74-9	9	4155
DB01	-10	10	3F33
DB02	-11	29	A65A
DB03	-13	31	9024
DB04	-14	32	DE6D
DB05	-15	11	D6FA
DB06	-16	12	7AC3
DB07	-17	13	0477
DB08	U85-9	9	A814
DB09	-10	10	C26B
DB10	-11	29	D909
DB11	-13	31	5FAA
DB12	-14	32	5925
DB13	-15	11	610D
DB14	-16	12	B8AB
DB15	-17	13	ADD3



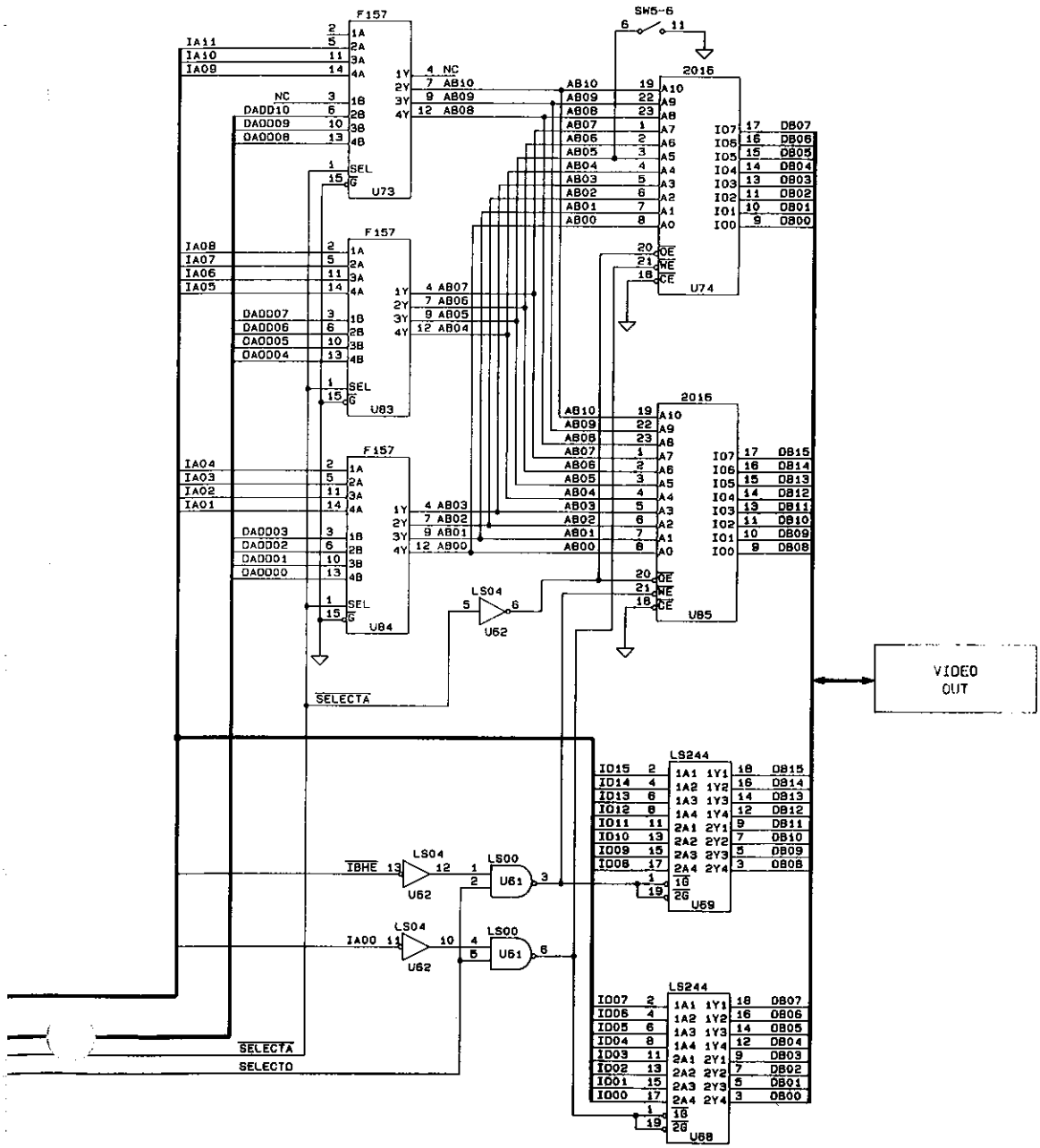


Figure 4-94: Video RAM Functional Test

Programmed Functional Test 4.9.5.

The *tst_vidram* program is the programmed functional test for the Video RAM functional block. This program checks the two RAM ICs U74 and U85 using the *gfi test* command. If the *gfi test* command fails, the *abort_test* program is executed and GFI troubleshooting begins. (See the Bus Buffer functional block for a discussion of the *abort_test* program).

```
program tst_vidram

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the VIDEO RAM functional block.                   !
!                                                                      !
! This program tests the VIDEO RAM functional block of the Demo/Trainer.!
! The gfi test command and I/O module are used to perform the test.    !
!                                                                      !
! TEST PROGRAMS CALLED:                                                !
!    abort_test (ref-pin)              If gfi has an accusation        !
!                                      display the accusation else     !
!                                      create a gfi hint for the       !
!                                      ref-pin and terminate the test!
!                                      program (GFI begins trouble-    !
!                                      shooting).                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Setup

   print "\nl\nlTESTING VIDEO RAM Circuit"

! Main part of Test

   podsetup 'enable ~ready' "on"

   if gfi test "U74-9" fails then abort_test("U74-9")
   if gfi test "U85-9" fails then abort_test("U85-9")

   print "VIDEO RAM TEST PASSES"
end program
```

Stimulus Programs and Responses 4.9.6.

Figure 4-95 is the stimulus program planning diagram for the Video RAM functional block. The *video_scan* stimulus program initializes video RAM by executing *video_fill*, which fills video RAM with characters including blinking characters. The *levels* stimulus program provides the appropriate stimuli to measure the asynchronous level of various outputs. The *video_rdy* stimulus

program stimulates the Video RAM Ready (VRAMRDY) generation circuit by writes made to the write-only video RAM.

All these stimulus programs (except *levels*) execute *video_init* before any measurements are made on the video circuitry.

Stimulus Program Planning

PROGRAM: LEVELS
MEASURES STATIC LEVELS
MEASUREMENT AT: U62-6 U61-3,6 U84-12,9,7,4 U83-12,9,7,4 U73-12,9,7 U69-3,5,7,9,12,14,16,18 U68-3,5,7,9,12,14,16,18

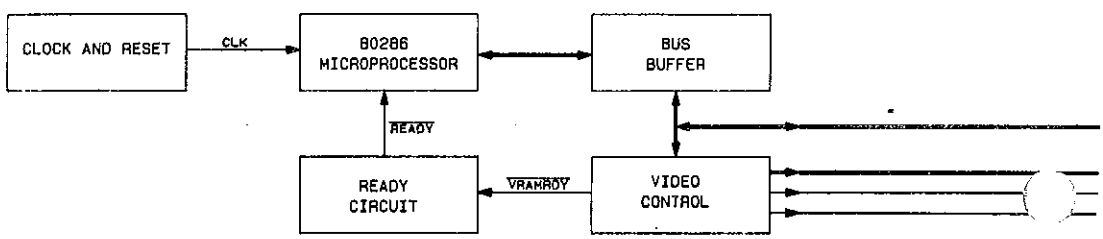
[redacted]

PROGRAM: VIDEO_SCAN
EXECUTES VIDEO_INIT, VIDEO_FIL1, AND MEASURES ALL CIRCUITRY WHERE DATA IS CLOCKED THROUGH BY CHARACTERS
[redacted]

INITIALIZATION PROGRAM: VIDEO_INIT
INITIALIZES VIDEO REGISTERS TO STANDARD OPERATING MODE
MEASUREMENT AT: (NONE)

INITIALIZATION PROGRAM: VIDEO_FIL1
INITIALIZES VIDEO RAM WITH BLINKING CHARACTERS
MEASUREMENT AT: (NONE)

INITIALIZATION PROGRAM: VIDEO_FIL2
INITIALIZES VIDEO RAM WITHOUT BLINKING CHARACTERS
MEASUREMENT AT: (NONE)



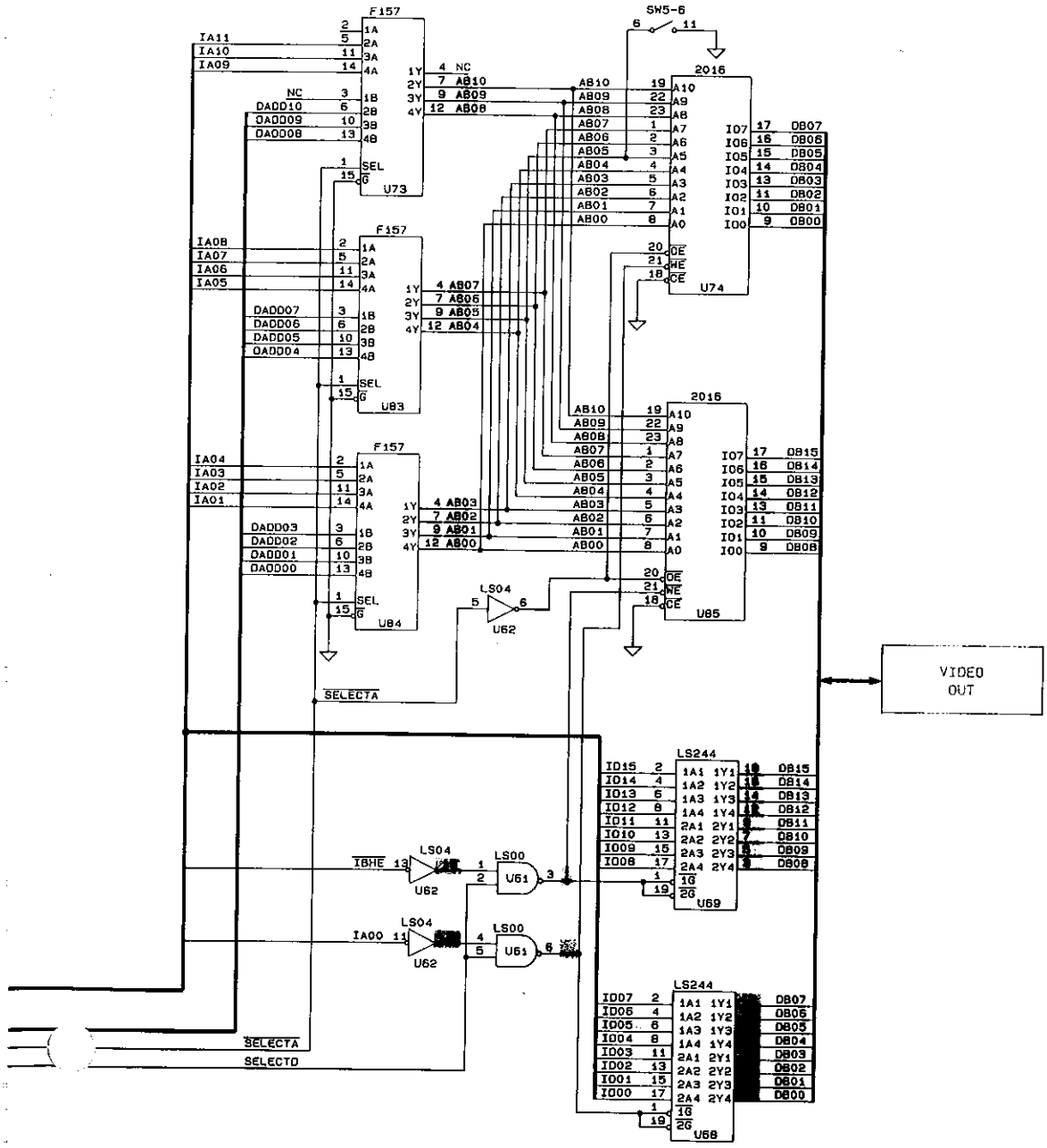


Figure 4-95: Video RAM Stimulus Program Planning

Summary of Complete Solution for Video RAM 4.9.7.

The entire set of programs and files needed to test and GFI troubleshoot the Video RAM functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Video RAM)

Programs (PROGRAM):

TST_VCTRL	Functional Test	Section 4.9.5
LEVELS	Stimulus Program	Figure 4-92
VIDEO_RDY	Stimulus Program	Figure 4-90
VIDEO_SCAN	Stimulus Program	Figure 4-77
VIDEO_INIT	Initialization Program	Figure 4-79
VIDEO_FIL1	Initialization Program	Figure 4-80
VIDEO_FIL2	Initialization Program	Figure 4-81

Stimulus Program Responses (RESPONSE):

LEVELS	Figure 4-93
VIDEO_RDY	Figure 4-91
VIDEO_SCAN	Figure 4-78

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

VID_FILL1	Initialization Data File
VID_FILL2	Initialization Data File

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

BUS BUFFER FUNCTIONAL BLOCK 4.10.

Buses and Bus Buffers 4.10.1.

In addition to the bus at the pins of a microprocessor, many microprocessor-based designs include additional internal buses connecting the microprocessor, memory, I/O devices, and other circuitry. These internal buses are often separated by buffers or latches which complicate testing and troubleshooting.

For purposes of testing and troubleshooting, a bus is a group of signals that operate in an identical manner, such as an address or data bus. The bus is a connection between a *sending* output and one or more *receiving* inputs. For internal buses, the sending and receiving components may be buffers. Buffers separate internal buses from the microprocessor bus. A fault such as two buffered address lines tied together cannot be directly detected from the microprocessor bus. Thus, some faults on the buffered bus may go undetected by the built-in BUS TEST.

The key to testing a bus is that while there may be multiple outputs to the bus, only one should be active at any time. Each bus has an associated set of control and status lines which must also be tested.

Considerations for Testing and Troubleshooting 4.10.2.

There are several methods of testing bus buffers. For buses, it is usually desirable to test all combinations of bus signal levels to verify that:

- All lines are drivable.
- No two lines are shorted together.
- No lines are open between the master and the receivers.

This is particularly desirable for data and address buses, whose lines are often physically adjacent. These lines may be subject to manufacturing defects and failure modes, such as:

- Over-etching of traces causing open lines.
- Under-etching of traces causing shorted lines.
- Solder bridges causing shorted lines.
- Faulty or damaged parts causing lines to be stuck or open.
- Faulty or damaged parts that have incorrect logical behavior.

Bus and Buffer Testing Capabilities

A ramp function is useful for testing buses and their associated buffers. The ramp function is a binary progression (i.e. a sequence of ascending numbers) covering all combinations of signal values. The ramp counts through all the values, starting at the lowest and ending at the highest. For large groups of signals, ramping over the full range can take considerable time. A means of ramping through a limited range by selecting a group of bits via a mask is therefore provided. For example, a 32-bit address bus may be covered by performing four ramping operations for each set of 8 bits (each group of which is probably associated with a particular buffer). This requires only 4x28 or 1024 operations vs. 2^{32} or 4.3 billion operations!

To troubleshoot a bus effectively, ramping operations must cover all normal transitions for logically adjacent lines. In the example above, suppose ramping operations covered address lines A0 through A7, A8 through A15, etc. If A7 and A8 are tied, the fault may not be discovered. It is therefore advisable to overlap ramping operations in order to provide the additional fault coverage. A portion of a TL/1 stimulus program might look like this:

```
rampaddr addr $F0000000, mask $1FF
rampaddr addr $F0000000, mask $1FF00
rampaddr addr $F0000000, mask $1FF0000
rampaddr addr $F0000000, mask $FF000000
```

There would be $3 \times 2^9 + 2^8$ or 1792 iterations vs. 1024 in the preceding example. Overlapping ramp functions usually takes little additional test time.

Several built-in ramp and toggle stimulus functions are available: In TL/1, the commands are *rampaddr, rampdata, toggleaddr, toggledata,* and *togglecontrol* (see Section 3 of the *TL/1 Reference Manual).* From the operator's keyboard, the STIM key provides these functions (see Section 5 of the *Technical User's Manual).*

As described earlier in Section 2.2, the 9100A/9105A can make five types of measurements to determine whether a node is good or bad. The list below describes how these five measurement types relate to bus buffers. The combination of *CRC signatures* and *asynchronous level history* is recommended for most bus node measurements, except when data buses are being measured.* Data buses are bidirectional and can be set to high-impedance levels between valid data times. In this case, CRC signatures with synchronous level history are the recommended measurement combination.

- *CRC signatures* are useful when associated with stimulus functions, since a unique signature results from a relatively large number of signal transitions. For a given stimulus program, two nodes that are tied will almost always have the same signature, different from the known-good signature.
- *Asynchronous level history* is useful when trying to determine whether a bus node is stuck. In this mode, the probe or I/O module will report all of a node's three states during the measurement period: logic 1, logic 0, or invalid X (high-impedence). Asynchronous level history is very useful for detecting glitches (short pulses) and is usually used together with CRC signatures. It should not, however, be used on data buses, which are bidirectional

and can be set to high impedance; since three-state conditions are not predictable on such lines, they may cause the measurement to fail. To measure data buses, use synchronous level history with CRC signatures.

- *Synchronous (clocked) level history* is used to measure signal levels at clock edges. This is useful for separation of signals present at the specified clock edges from signals present at other times. Clocked level history reports logic states in the same way as asynchronous level history. Measure data buses with this method, using the stable clock to avoid the three-state condition.
- *Transition count* is used in place of CRC signatures when there is no stable clock available.
- *Frequency* can be used to measure periodic bus cycles, such as refresh, or to verify the frequency of system clocks.

Address Buffers

When troubleshooting address buffers, the physical address map of the UUT can be used to partition address buffer tests. For example, a set of address lines may be part of the I/O memory and associated with a particular buffer. Thus, a *rampaddr* command over the specific I/O memory range may be sufficient to verify proper operation of the buffer.

Other examples of address-bus partitions are:

- *Mapped address lines* are the microprocessor address lines that are translated or mapped into another set of lines by a fast RAM or VLSI component.
- *System bus address lines* are the address lines (usually different from the microprocessor address lines) in the system bus. These are usually buffered independently from internal address lines.
- *Internal (local) address lines* are usually buffered separately for local memory or other components.

Address lines may be latched as well as buffered. In latched applications, the latch acts as a buffer and should therefore be included in the Bus Buffer functional block.

Data Buffers

Many UUTs with 16- and 32-bit microprocessors and standard buses have separate buffers for each group of eight bits with three-state and direction-control lines that can be controlled independently. There may also be buffers that allow swapping or repositioning of bytes within a word. The *rampdata* command, combined with CRC signatures, can be used to diagnose data-bus-related errors in a similar way as *rampaddr*.

The *rampdata* command is a stimulus with the microprocessor as the node source. To apply stimulus in the opposite direction, read data from a component on the bus (such as RAM, ROM or DMA). To do this, write a stimulus program to read data from the component, and record signatures in the same way as for *rampdata*. A ROM is a convenient component since, once programmed, it contains a pseudo-random pattern which, over a given address range, will generate meaningful signatures for the individual data lines. There is usually a ROM associated with each byte of the data bus. The *read* or *rampaddr* commands will provide the addresses for generating the data to be read from the ROM.

Control Buffers

Control lines may sometimes be generated by an LSI component associated with the microprocessor. The LSI component is included here in the bus buffer functional block because it performs a function similar to the bus buffer. The testing and troubleshooting of these components proceeds as though they were simple buffers.

A faulty control buffer can cause the address-bus and data-bus tests to fail. Control signals are tested by performing reads and

writes in all possible address spaces and all possible data widths. Some control signals can be tested by the *togglecontrol* command. The control buffers should be checked as the control lines are stimulated.

Several types of control lines present problems. Here are some general guidelines:

- *Bus exchange* signals are used to relinquish control of the microprocessor bus to another master. Large systems may have a bus arbitration circuit for granting the bus to a requesting component. These circuits should probably be treated separately from the bus buffer block. Asynchronous access to the bus during tests should be restricted and access should be limited to the specific master acting as the stimulus source. The state of the bus-request line can be determined with the measurement techniques described above.
- *Direction control* signals control the direction of data flow through the buffers and are usually connected directly to inputs on the buffer ICs. These signals may be derived from microprocessor status lines, LSI components, or buffered versions of the microprocessor signal. There may also be separate read and write signals for different physical memory or I/O address spaces. The logic state for each of these signals should be verified for the appropriate bus cycle.
- *Wait-state control* signals such as READY on the 80286 microprocessor and ~DTACK on the 68000 microprocessor extend the bus cycle to accommodate slower components. Stuck wait-state control lines will cause bus-related functions to fail. If the pod is the stimulus source, a stuck high (negated) condition on Ready will cause a pod timeout. When the pod timeout occurs, a message like "enabled line ~READY PIN 63 causes timeout" (when using an 80286 pad) will result. The line can be disabled and testing can proceed. For example, when a ROM requiring one wait state is the stimulus source and the Ready or ~DTACK signal is stuck low (asserted), the bus cycles may be completed but bad

data may be produced. As with other control lines, asynchronous level history is useful in detecting stuck control lines.

- *Reset* is a system-wide control signal which may be included in the bus buffer functional block. A reset signal stuck in the asserted state will probably affect many tests. Often, the only way to verify operation of a Reset signal without cycling the power on the UUT is to externally assert the signal using a switch, or overdriving device such as the probe. The various nodes which distribute the reset signal via buffers may be verified using the asynchronous level history measurement.

Miscellaneous Lines

System clocks are sometimes associated with the control lines for a particular bus. These clocks are often used to synchronize external events with a bus cycle, they are often an integral part of control-signal generation, and they can cause control-signal faults if they are faulty.

Clocks asynchronous with the microprocessor clock are sometimes used to run state machines associated with bus- and buffer-control circuitry. Nodes that distribute these clocks via buffers can be measured with the probe or I/O module programmed to measure frequency. There is no stimulus associated with these frequency measurements, even though a stimulus program is used to set the mode on the measurement device. The same is true for the program used to measure asynchronous levels. These programs are referred to as response-only stimulus programs. See the *levels* and *frequency* programs in Section 4.8.6 and 4.12.6.

Pull-up or pull-down resistors which establish static logic levels on buses when there are no active outputs should also be tested. Levels can be verified with asynchronous level history measurements.

VLSI Components

Some VLSI components integrate a large amount of peripheral microprocessor circuitry associated with personal computer designs, including the bus buffers. Operation of these components can be quite complex. To simplify stimulus program design, the buffer portion of these components, along with the associated control signals, can be grouped in a separate functional block from the other functions of the component. Testing can then be done in a manner similar to that for discrete buffers.

Connectors

Connectors are a part of many bus buffering functional blocks. Whether these are test connectors, card-edge connectors or sockets, they are components that can cause stuck, tied, or open bus lines. Connectors should therefore be included in tests.

Bus Buffer Example 4.10.3.

The bus buffer in the Demo/Trainer UUT, Figure 4-96, uses an 82288 bus controller (U15) to decode status lines ~S0, ~S1, M/~IO from the microprocessor and to generate command signals for bus-cycle control. An "I" is appended to some mnemonics, signifying internal (buffered) signals. For example, data-bus lines D00-D15 become internal lines ID00-15.

The address bus (A00-23) is buffered with latches U2, U16, and U22. The rising edge of each ALE transition latches a new address.

For the data bus (D00-15), the 82288 outputs control signals DEN (data enable) and DT/~R (data transmit/receive). These two signals control the state of data-bus transceivers U23 and U3. For write cycles, both DEN and DT/~R are high. For read cycles, DEN is high and DT/~R is low.

Keystroke Functional Test 4.10.4.

Part A:

Use a 20-pin clip module on side A of I/O module 1 to test data and control outputs from the microprocessor. Use the SYNC, I/O MOD, and STIM keys with the commands below for each of the following parts: U3, U23, U22, U15, and U45. The correct measurement for each pin is shown in the response table in Figure 4-96.

```
SYNC I/O MOD 1 TO POD DATA
ARM I/O MOD 1 FOR CAPTURE USING SYNC
RAMP DATA 0 MASKED BY FF, ADDR 0
... (ADDR OPTION: I/O BYTE)
RAMP DATA 0 MASKED BY FF00, ADDR 0
... (ADDR OPTION: MEMORY WORD)
SHOW I/O MOD 1 PIN <see response table> ...
... CAPTURED RESPONSES
```

NOTE

The SHOW command requires a clip module pin number rather than a part pin number. This requires you to translate part pin numbers to clip module pin numbers (see Appendix B of the Technical User's Manual). For your convenience, this translation has been done for you in this example, and the results are shown in the "I/O MOD PIN" column of the response table in Figure 4-96.

Part B:

Use a 20-pin clip module on side A of I/O module 1 to test data input to the microprocessor from the ROMs. Use the SYNC, I/O MOD, and STIM keys with the commands below for U3 and then for U23. The correct measurement for each pin is shown in the response table in Figure 4-97.

```
SYNC I/O MOD 1 TO POD DATA
ARM I/O MOD 1 FOR CAPTURE USING SYNC
RAMP ADDR E0000 MASKED BY 1FE
... (ADDR OPTION: MEMORY WORD)
SHOW I/O MOD 1 PIN <see response table> ...
... CAPTURED RESPONSES
```

Part C:

1. Use a 20-pin clip module on side A of I/O module 1 to test addresses and control outputs from the microprocessor.

2. Use the SETUP MENU key with the following commands to disable Ready so all addresses can be generated:

```
SETUP POD ENABLE ~READY OFF
SETUP POD REPORT FORCING SIGNALS OFF
```

3. Use the SYNC, I/O MOD, and STIM keys with the commands below for each of the following parts: U16, U2, U22, U15, and U45. The correct measurement for each pin is shown in response table #1 in Figure 4-98.

```
SYNC I/O MOD 1 TO POD ADDR
ARM I/O MOD 1 FOR CAPTURE USING SYNC
RAMP ADDR 0 MASKED BY FFC00
... (ADDR OPTION: MEMORY WORD)
RAMP ADDR 0 MASKED BY 7FF
... (ADDR OPTION: I/O BYTE)
SHOW I/O MOD 1 PIN <see response table> ...
... CAPTURED RESPONSES
```

4. Use the SYNC, I/O MOD, and STIM keys with the commands below for part U15. The correct measurement for each pin is shown in response table #2 in Figure 4-98.

```
SYNC I/O MOD 1 TO POD DATA
     (Note that this is pod DATA sync)
ARM I/O MOD 1 FOR CAPTURE USING SYNC
RAMP ADDR 0 MASKED BY FFC00
... (ADDR OPTION: MEMORY WORD)
RAMP ADDR 0 MASKED BY 7FF
```

```
... (ADDR OPTION: I/O BYTE)
SHOW I/O MOD 1 PIN 8 CAPTURED RESPONSES
SHOW I/O MOD 1 PIN 12 CAPTURED RESPONSES
```

5. After completing this functional test, use the SETUP MENU key with the commands below to enable Ready and to restore reporting of active forcing signals.

```
SETUP POD ENABLE READY ON
SETUP POD REPORT FORCING SIGNALS ON
```

Part D:

Use a 20-pin clip module on side A of I/O module 1 to test control outputs during interrupt acknowledge by using the pod program named FRC_INT. Use the SETUP MENU, SYNC, and I/O MOD keys with the commands below for U15 and then for U45. The correct measurement for each pin is shown in the response table in Figure 4-99.

```
SETUP POD ENABLE ~READY ON
SETUP POD REPORT FORCING SIGNALS ON
SETUP POD INTA_ACK ON
SYNC I/O MOD 1 TO POD INTA
ARM I/O MOD 1 FOR CAPTURE USING SYNC
POD: FRC_INT
... (ADDR OPTION: MEMORY WORD)
SHOW I/O MOD 1 PIN <see response table> ...
... CAPTURED RESPONSES
```

Keystroke Functional Test (Part A)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	I/O MOD U3 U15 U23 U45 U22

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
ID00	U3-11	11	A A 6 1
ID01	-12	12	9 9 D F
ID02	-13	13	8 7 9 3
ID03	-14	14	E 6 1 8
ID04	-15	15	F 5 1 3
ID05	-16	16	4 F F B
ID06	-17	17	3 6 0 0
ID07	-18	18	B 2 5 9
ID08	U23-11		9 6 E C
ID09	-12	12	7 2 5 B
ID10	-13	13	E 5 E D
ID11	-14	14	5 B E 0
ID12	-15	15	7 E 2 5
ID13	-16	16	8 5 E A
ID14	-17	17	7 7 C 7
ID15	-18	18	6 E B E
ICOD/INTA	U22-5	5	F E A 2
IM/IO	-6	6	B B 3 4
WRITE	U15-9	9	F E A 2
IWRITE	-11	11	B B 3 4
DEN	-16	16	4 5 9 6
ALATCH	U45-8	14	4 5 9 6

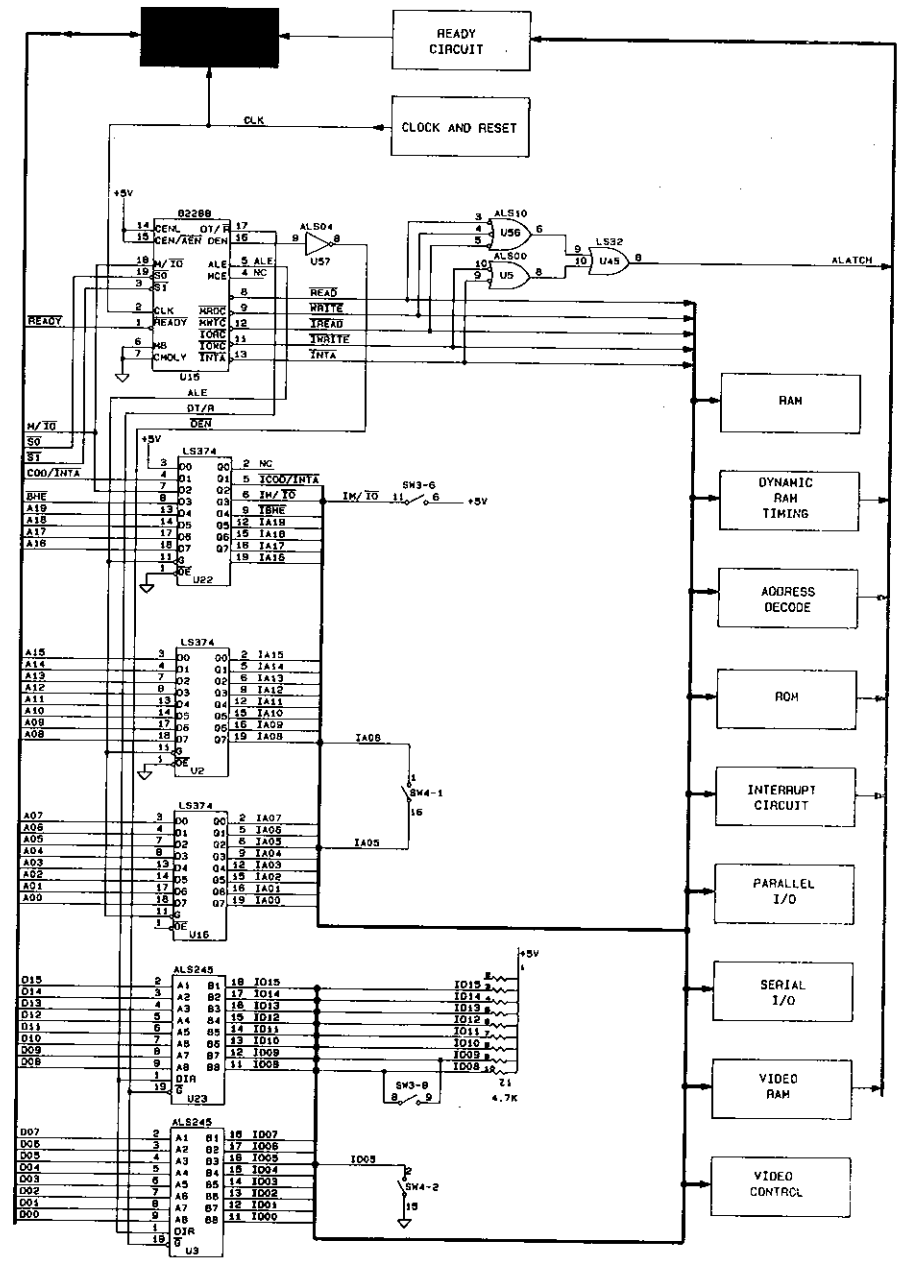


Figure 4-96: Bus Buffer Functional Test (Part A)

Keystroke Functional Test (Part B)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	I/O MOD U3 U23

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
D00	U3-9	9	4 5 D D
D01	-8	8	C F 8 3
D02	-7	7	B D 7 9
D03	-6	6	8 A 7 6
D04	-5	5	6 6 F 3
D05	-4	4	F A B 5
D06	-3	3	5 3 4 E
D07	-2	2	8 D 0 A
D08	U23-9	9	7 3 E 9
D09	-8	8	A C 8 4
D10	-7	7	5 0 B B
D11	-6	6	5 B 3 B
D12	-5	5	0 6 E F
D13	-4	4	0 0 A 0
D14	-3	3	6 B F 0
D15	-2	2	5 2 E E

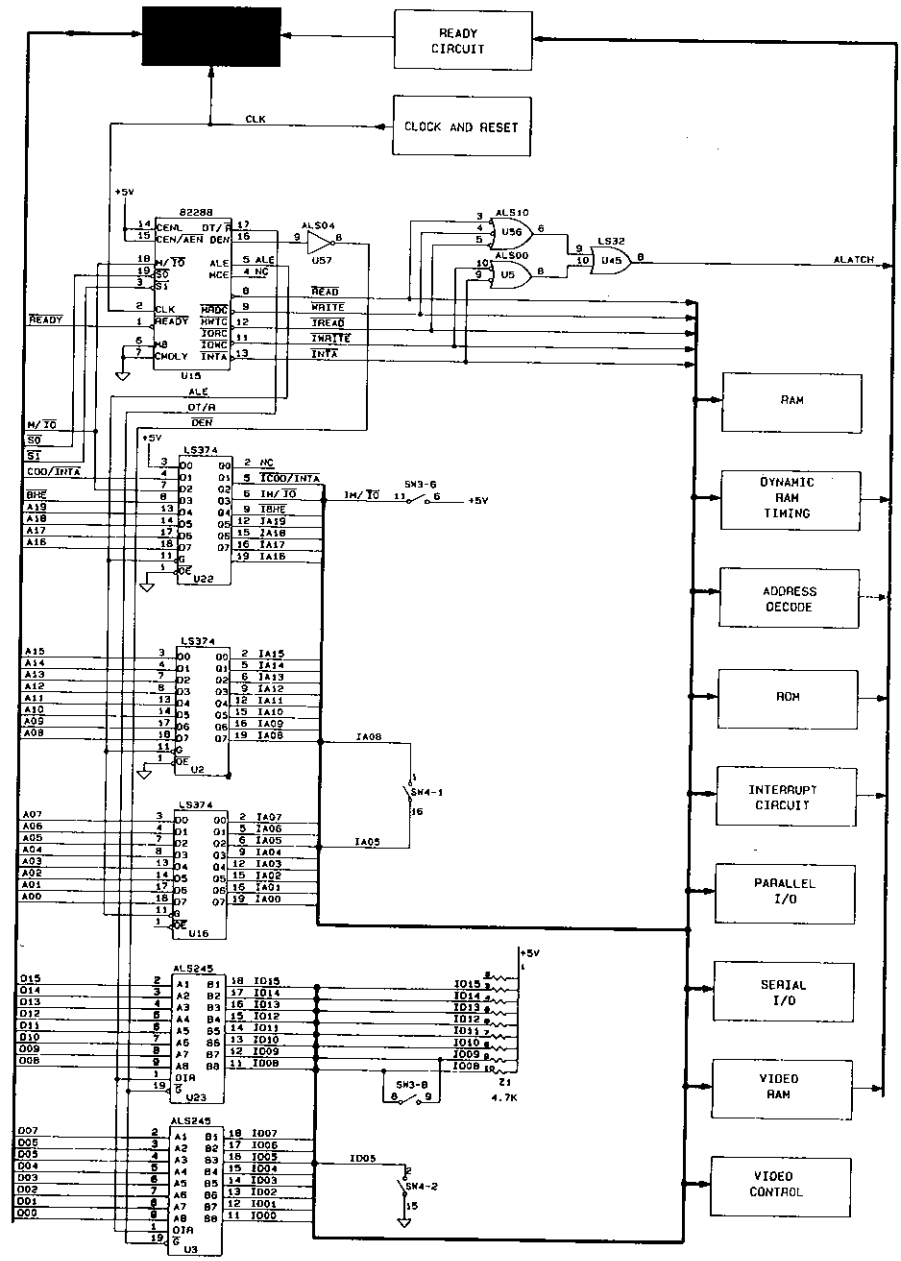


Figure 4-97: Bus Buffer Functional Test (Part B)

Keystroke Functional Test (Part C)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	I/O MOD U16 U15 U2 U45 U22

RESPONSE TABLE #1

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE	ASYNC LEVEL
IA00	U16-19	19	D 6 2 F	
IA01	-16	16	B 2 1 A	
IA02	-15	15	D A 0 7	
IA03	-12	12	1 0 2 4	
IA04	-9	9	E 0 3 0	
IA05	-6	6	4 4 4 2	
IA06	-5	5	4 F 2 A	
IA07	-2	2	0 7 7 2	
IA08	U2-19	19	9 6 3 5	
IA09	-16	16	1 7 3 4	
IA10	-15	15	4 1 B A	
IA11	-12	12	4 5 5 A	
IA12	-9	9	B E E D	
IA13	-6	6	3 8 6 8	
IA14	-5	5	D E E 6	
IA15	-2	2	C 2 6 5	
IA16	U22-19	19	F 8 A 7	
IA17	-16	16	9 1 9 6	
IA18	-15	15	2 5 E 4	
A19	-12	12	4 A 7 4	
IBHE	-9	9	A C 5 F	
ALE	U15-5	5	2 A 4 2	1 0
INTA	-13	5	2 A 4 2	1
DT/R	-17	17	9 D 9 2	1 0
ALATCH	U45-8	14	B 7 D 0	1 0

RESPONSE TABLE #2

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
READ	U15-8	8	[redacted]
IREAD	-12	12	[redacted]

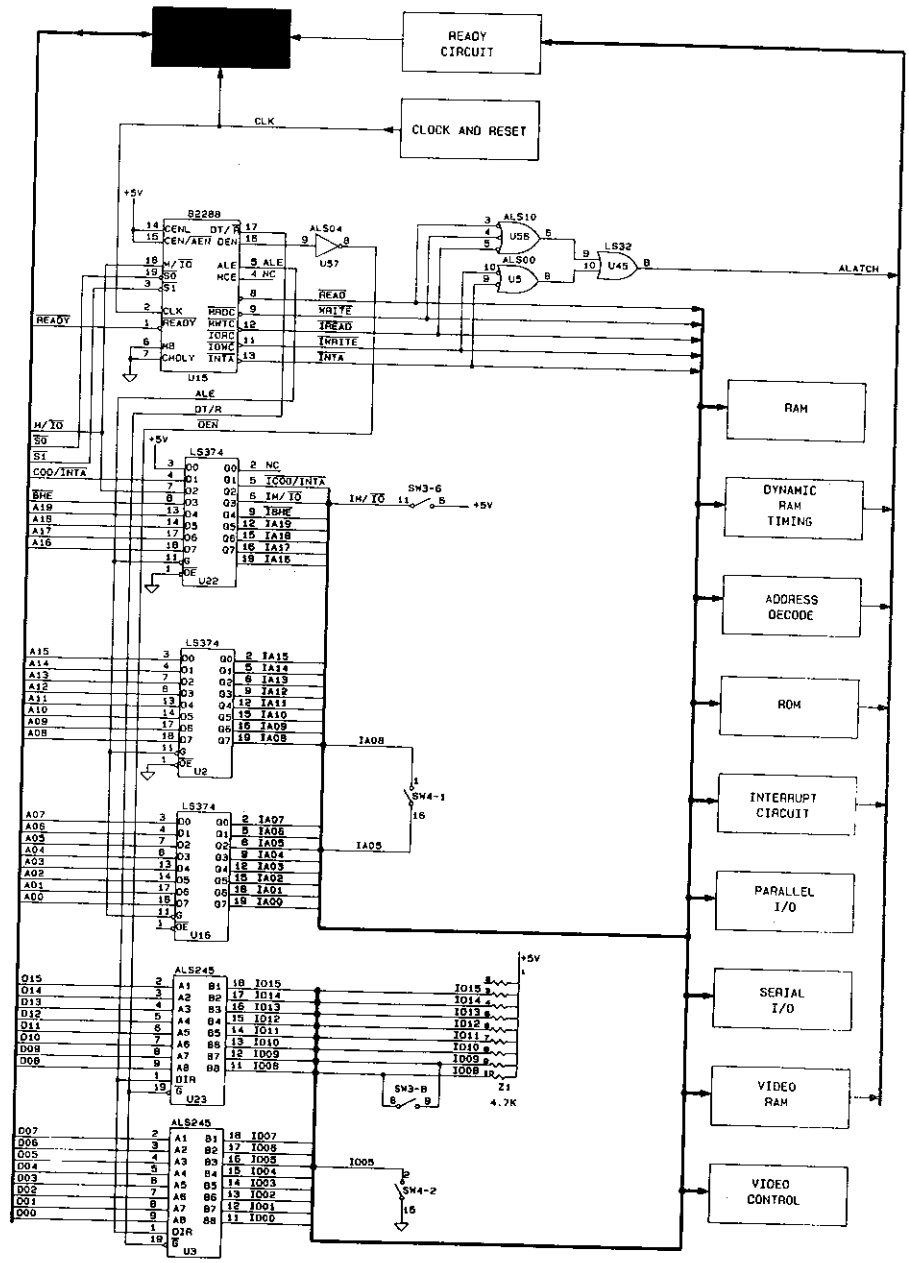


Figure 4-98: Bus Buffer Functional Test (Part C)

Keystroke Functional Test (Part D)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	I/O MOD U15 U45 U22

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
ALE	U15-5	5	0 0 0 0
READ	-8	8	0 0 0 1
WRITE	-9	9	0 0 0 1
IWRITE	-11	11	0 0 0 1
IREAD	-12	12	0 0 0 1
INTA	-13	13	0 0 0 0
DT/R	-17	17	0 0 0 0
ALATCH	U45-8	14	0 0 0 1
ICOD/INTA	U22-5	5	0 0 0 0

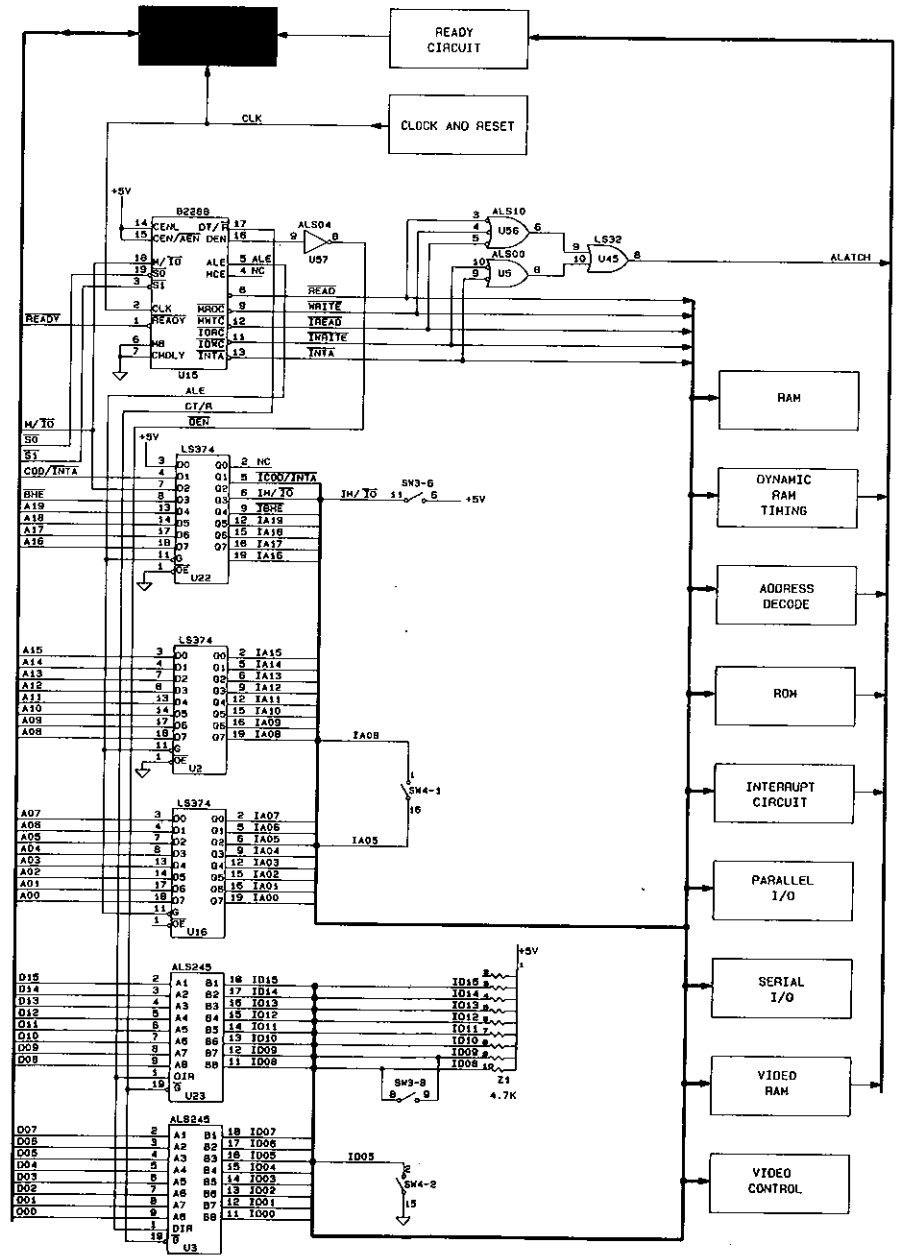


Figure 4-99: Bus Buffer Functional Test (Part D)

Programmed Functional Test 4.10.5.

The *tst_buffer* program is the programmed functional test for the Bus Buffer functional block. The *gfi test* command is used to run all stimulus programs defined for the parts tested and to compare the results against known-good responses stored in the response files. If all results are good, the *gfi test* passes; otherwise the *gfi test* fails.

The *tst_buffer* program performs a *gfi test* on address buffer U16. If the *gfi test* fails, a program called *abort_test* is executed using a variable containing the part and pin number that was tested by the *gfi test* command. A listing for the *abort_test* program is included in Appendix C.

The *abort_test* program uses the *gfi accuse* command to see if an accusation exists. If there is no accusation, a *gfi hint* containing the part number and pin number is generated and the program is terminated (the *gfi hint* gives GFI a place to start troubleshooting). If an accusation does exist, the *abort_test* program displays the accusation and the program is terminated.

The *gfi test* (and execution of *abort_test* if the *gfi test* fails) is repeated for the other two address buffers U2 and U22 and then for the data bus buffers U3 and U23.

```
program tst_buffer

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the BUS BUFFER functional block.                     !
!                                                                         !
! This program tests the BUS BUFFER functional block of the               !
! Demo/Trainer.  The gfi test command and I/O module are used to clip     !
! over the buffers and perform the test.                                  !
!                                                                         !
! TEST PROGRAMS CALLED:                                                   !
!    abort_test (ref-pin)                 If gfi has an accusation        !
!                                         display the accusation else     !
!                                         create a gfi hint for the       !
!                                         ref-pin and terminate the test!
!                                         program (GFI begins trouble-    !
!                                         shooting).                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
   print "\n1TESTING BUS BUFFER Circuit"

! Test ADDRESS BUS

   if gfi test "U16-1" fails then abort_test("U16-1")
   if gfi test "U2-1" fails then abort_test("U2-1")
   if gfi test "U22-1" fails then abort_test("U22-1")

! Test DATA BUS

   if gfi test "U3-1" fails then abort_test("U3-1")
   if gfi test "U23-1" fails then abort_test("U23-1")

   print "BUS BUFFER TEST PASSES"
end program
```

Stimulus Programs and Responses 4.10.6.

Figure 4-100 is the stimulus program planning diagram for the Bus Buffer functional block.

The stimulus programs *addr_out, ctrl_out1, ctrl_out2, ctrl_out3,* and *data_out* exercise outputs going outward from the microprocessor. The *rom1_data* stimulus program stimulates the outputs of the data buffers that are connected to the microprocessor.

Stimulus Program Planning

PROGRAM: ADDR_OUT
EXERCISES ALL ADDRESS LINES FROM THE MICROPROCESSOR
MEASUREMENT AT: U16-19,16,15,12,9,6,5,2 U2-19,16,15,12,9,6,5,2 U22-19,16,15,12,9

[redacted]

PROGRAM: DATA_OUT
EXERCISES ALL DATA LINES AS OUTPUTS FROM THE MICROPROCESSOR
MEASUREMENT AT: U3-11,12,13,14,15,16,17,18 U23-11,12,13,14,15,16,17,18

[redacted]

[redacted]

[redacted]

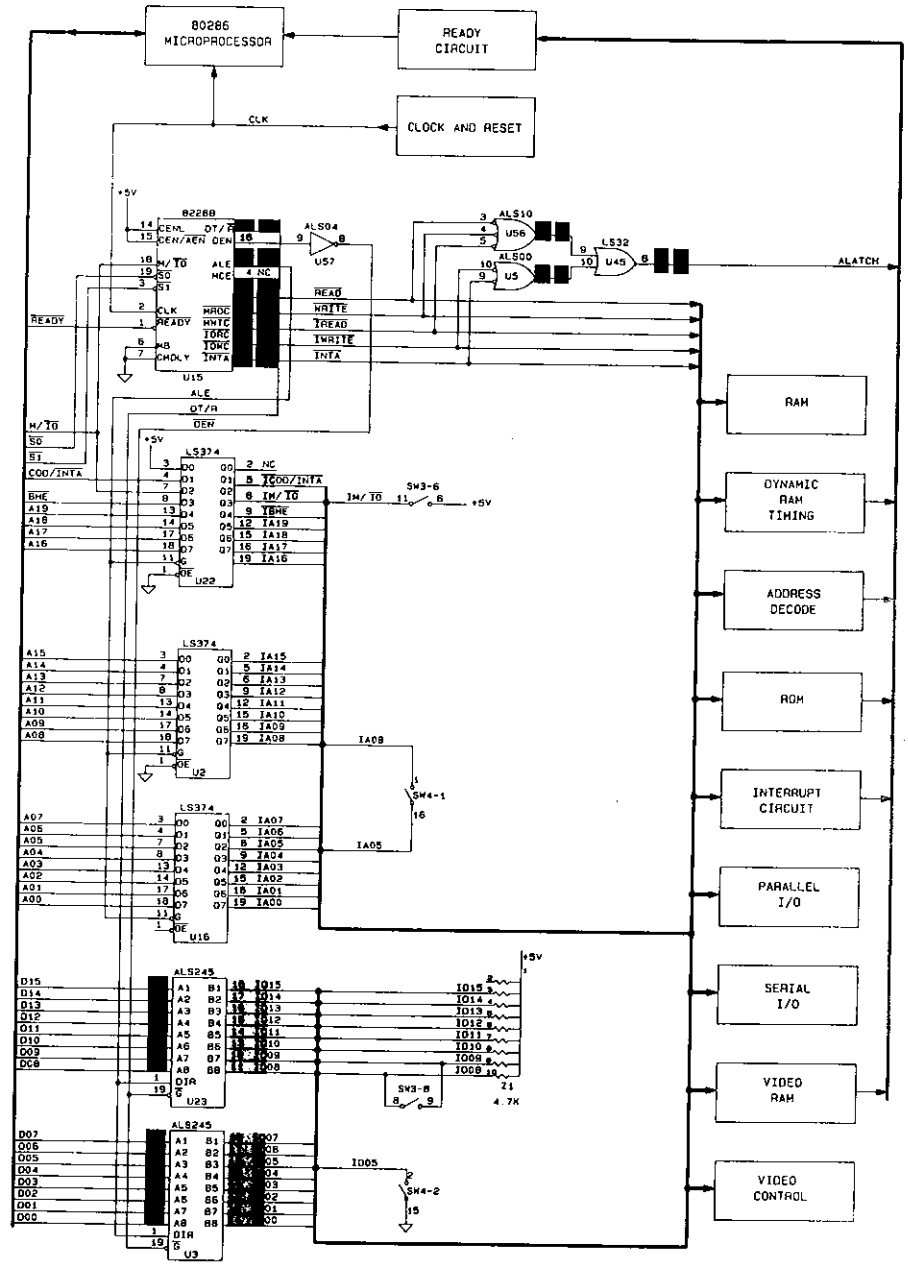


Figure 4-100: Bus Buffer Stimulus Program Planning

```
program ctrl_out2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM for bus controller, U15 & uP ctrl lines.             !
!                                                                        !
! Stimulus programs and response files are used by GFI to backtrace     !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                        !
! This stimulus program is one of the programs which creates activity   !
! in the kernel area of the UUT.  These programs create activity with   !
! or without the ready circuit working properly.  Because of this, all  !
! the stimulus programs in the kernel area must disable the READY input !
! to the pod, then perform the stimulus, then re-enable the READY input !
! to the pod.  The 80286 microprocessor has a separate bus controller;  !
! for this reason, disabling ready and performing stimulus can get the  !
! bus controller out of synchronization with the pod.  Two fault        !
! handlers trap pod timeout conditions that indicate the bus controller !
! is out of synchronization.  The recover() program is executed to      !
! resynchronize the bus controller and the pod.                         !
!                                                                        !
! TEST PROGRAMS CALLED:                                                  !
!    recover     ()                       The 80286 microprocessor has a!
!                                         bus controller that is totally!
!                                         separate from the pod.  In    !
!                                         some cases the pod can get out!
!                                         of sync with the bus control- !
!                                         ler.  The recover program     !
!                                         resynchronizes the pod and the!
!                                         bus controller.               !
!                                                                        !
! GRAPHICS PROGRAMS CALLED:                                              !
!    (none)                                                              !
!                                                                        !
! Local Variables Modified:                                              !
!    devname                              Measurement device            !
!    io_byte                              I/O BYTE address space        !
!    mem_word                             MEMORY WORD address space     !
!                                                                        !
! Global Variables Modified:                                             !
!    recover_times                        Reset to Zero                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-101: Stimulus Program *(ctrl_out2)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle
handle pod_timeout_no_clk
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program CTRL_OUT2"

! Set addressing mode and setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   io_byte = getspace space "i/o", size "byte"
   mem_word = getspace space "memory", size "word"
   reset device devname
   sync device devname,mode "pod"
   sync device "/pod", mode "data"
   old_cal = getoffset device devname
   setoffset device devname, offset (1000000 - 70)

! Present stimulus to UUT.

   arm device devname          ! Start response capture.
      setspace (mem_word)
      rampaddr addr $E0000, mask $1E
      rampdata addr $50000, data 0, mask $F
      setspace (io_byte)
      rampaddr addr 0, mask $5F00
      rampdata addr $2000, data 0, mask $F
   readout device devname      ! End response capture.

   setoffset device devname, offset old_cal
   podsetup 'enable ~ready' "on"
end program
```

Figure 4-101: Stimulus Program *(ctrl_out2) - continued*

```
STIMULUS PROGRAM NAME: CTRL_OUT2
DESCRIPTION:                                        SIZE:              261 BYTES

                  ------------------- Response Data ---------------------
   Node            Learned                Async Clk Counter                    Priority
Signal Src          With        SIG         LVL  LVL  Mode     Counter Range     Pin

U15-5          I/O MODULE   0000     1 0      0 TRANS
U15-8          PROBE        A5B1     1 0        TRANS
U15-8          I/O MODULE   A5B1     1 0        TRANS
U15-9          PROBE        A841     1 0        TRANS
U15-9          I/O MODULE   A841     1 0        TRANS
U15-11         PROBE        448E     1 0        TRANS
U15-11         I/O MODULE   448E     1 0        TRANS
U15-12         PROBE        F383     1 0        TRANS
U15-12         I/O MODULE   F383     1 0        TRANS
U15-13         I/O MODULE   BAFD     1          TRANS
U15-17         I/O MODULE   ECCF     1 0        TRANS
U5-8           I/O MODULE   FE73     1 0        TRANS
U45-8          I/O MODULE   BAFD     1 0        TRANS
U56-6          PROBE        448E     1 0        TRANS
U56-6          I/O MODULE   448E     1 0        TRANS
```

Figure 4-102: Response File *(ctrl_out2)*

```
program ctrl_out3

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM for bus controller, U15 & uP ctrl lines.                   !
!                                                                             !
! Stimulus programs and response files are used by GFI to backtrace           !
! from a failing node.  The stimulus program must create repeatable UUT       !
! activity and the response file contains the known-good responses for       !
! the outputs in the UUT that are stimulated by the stimulus program.         !
!                                                                             !
! This stimulus program is one of the programs which creates activity         !
! in the kernel area of the UUT.  These programs create activity with         !
! or without the ready circuit working properly.  Because of this, all        !
! the stimulus programs in the kernel area must disable the READY input       !
! to the pod, then perform the stimulus, then re-enable the READY input       !
! to the pod.  The 80286 microprocessor has a separate bus controller;        !
! for this reason, disabling ready and performing stimulus can get the        !
! bus controller out of synchronization with the pod.  Two fault              !
! handlers trap pod timeout conditions that indicate the bus controller       !
! is out of synchronization.  The recover() program is executed to            !
! resynchronize the bus controller and the pod.                               !
!                                                                             !
! TEST PROGRAMS CALLED:                                                       !
!    recover     ()                         The 80286 microprocessor has a!
!                                           bus controller that is totally!
!                                           separate from the pod.  In    !
!                                           some cases the pod can get out!
!                                           of sync with the bus control- !
!                                           ler.  The recover program     !
!                                           resynchronizes the pod and the!
!                                           bus controller.               !
!                                                                         !
!                                                                         !
!    frc_int     ()                         Pod program to Force Interrupt!
!                                           Ack.                          !
!                                                                         !
! GRAPHICS PROGRAMS CALLED:                                               !
!    (none)                                                               !
!                                                                         !
! Local Variables Modified:                                               !
!    devname                                Measurement device            !
!    io_byte                                I/O BYTE address space        !
!    mem_word                               MEMORY WORD address space     !
!                                                                         !
! Global Variables Modified:                                              !
!    recover_times                          Reset to Zero                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

(continued on the next page)

Figure 4-103: Stimulus Program *(ctrl_out3)*

```
! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
       devname = gfi device
   else
       devname = clip ref "U15"
   end if
   print "Stimulus Program CTRL_OUT3"

! Set addressing mode and setup measurement device.

   io_byte = getspace space "i/o", size "byte"
   mem_word = getspace space "memory", size "word"
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   reset device devname
   podsetup 'intr_ack on'
   sync device "/pod", mode "inta"
   sync device devname, mode "pod"

! Present stimulus to UUT.

   arm device devname          ! Start response capture.
      execute frc_int()        !   Force Interrupt Ack.
   readout device devname      ! End response capture.

end program
```

Figure 4-103: Stimulus Program *(ctrl_out3) - continued*

STIMULUS PROGRAM NAME: CTRL_OUT3
DESCRIPTION: SIZE: 282 BYTES

Node Signal Src	Learned With	Response Data SIG	Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U15-5	I/O MODULE	0000	1	0	TRANS		
U15-8	PROBE	0001	1	0	TRANS		
U15-8	I/O MODULE	0001	1	0	TRANS		
U15-9	PROBE	0001	1		TRANS		
U15-9	I/O MODULE	0001	1		TRANS		
U15-11	PROBE	0001	1		TRANS		
U15-11	I/O MODULE	0001	1		TRANS		
U15-12	PROBE	0001	1		TRANS		
U15-12	I/O MODULE	0001	1		TRANS		
U15-13	I/O MODULE	0000	1	0	TRANS		
U15-17	I/O MODULE	0000	1	0	TRANS		
U4-3	I/O MODULE	0000	1	0	TRANS		
U5-8	I/O MODULE	0001	1	0	TRANS		
U45-8	I/O MODULE	0001	1	0	TRANS		
U56-6	PROBE	0000	1	0	TRANS		
U56-6	I/O MODULE	0000	1	0	TRANS		
U15-4	I/O MODULE	0000	1	0	TRANS		

Figure 4-104: Response File *(ctrl_out3)*

Summary of Complete Solution for Bus Buffer 4.10.7.

The entire set of programs and files needed to test and GFI troubleshoot the Bus Buffer functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Bus Buffer)

Programs (PROGRAM):

TST_BUFFER	Functional test	Section 4.10.5
ADDR_OUT	Stimulus Program	Figure 4-4
DATA_OUT	Stimulus Program	Figure 4-6
CTRL_OUT1	Stimulus Program	Figure 4-8
CTRL_OUT2	Stimulus Program	Figure 4-101
CTRL_OUT3	Stimulus Program	Figure 4-103
ROM1_DATA	Stimulus Program	Figure 4-16

Stimulus Program Responses (RESPONSE):

ADDR_OUT	Figure 4-5
DATA_OUT	Figure 4-7
CTRL_OUT1	Figure 4-9
CTRL_OUT2	Figure 4-102
CTRL_OUT3	Figure 4-104
ROM1_DATA	Figure 4-17

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

ADDRESS DECODE FUNCTIONAL BLOCK 4.11.

Introduction to Address Decode Circuits 4.11.1.

The Address Decode circuit of a UUT typically consists of the decoder ICs, an address path from the microprocessor to the decoder ICs, and the decoder outputs that select the peripheral devices on the UUT. Figure 4-105 shows such a circuit.

Many microprocessor systems contain an address latch or a buffer between the microprocessor and the address decoder ICs. The decoder ICs generally contain combinatorial logic that asserts one and only one of the decoder outputs for a given range of addresses. The address decoder typically has one or more enable input pins. The signals feeding these pins may be address lines or outputs from other decoders.

In some cases, the address decode logic is just one part of an LSI chip. In this situation, the LSI component should be partitioned so that only those inputs and outputs that relate to address decoding are part of the Address Decode functional block.

Considerations for Testing and Troubleshooting 4.11.2.

Use the 9100A/9105A's I/O module to test address decoding circuits. The general procedure is to characterize all decoder ICs and paths in the address decoding circuit of a known-good UUT, and then perform the same procedures on the suspect UUT, comparing results.

For each decoder IC in the circuit, the following test procedure can be used from the operator's keypad:

1. Clip the I/O module onto the IC.
2. Synchronize and arm the I/O module (see the *Technical User's Manual* for this procedure).

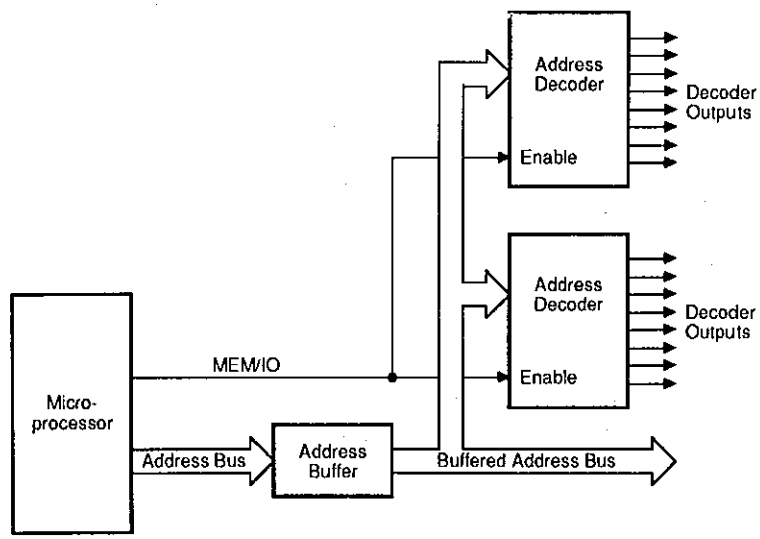


Figure 4-105: Typical Address Decode Functional Block

3. Run a stimulus procedure to make each output go high and low.

4. Use the SHOW I/O MOD command on the I/O MOD key (operator's keypad) to observe signatures on each pin of the IC.

5. Write down the signatures gathered from each pin on the IC, both inputs and outputs.

Since decoder outputs are typically asserted only over a specific address range, your stimulus procedure should also perform its reads and writes within that range for each decoder output. For example, consider a decoder with eight outputs, as follows:

Decoder Output	*Address Range (hex)*
~Y0	0-7FF
~Y1	800-FFF
~Y2	1000-17FF
~Y3	1800-2FFF
~Y4	3000-37FF
~Y5	3800-3FFF
~Y6	4000-47FF
~Y7	4800-4FFF

A stimulus procedure to test the first output, ~Y0, might consist of the following:

```
READ ADDR 0
READ ADDR 7FF
```

This will test the end points of the valid address range for ~Y0, to verify that ~Y0 is asserted (low) within that range. The same pair of reads within the valid address range of ~Y1 will test that ~Y0 is not asserted (high) outside the valid address range of ~Y0. You can use this strategy to test all of the decoder outputs with only 16 read operations.

If the outputs of a decoder IC are bad and the inputs are good, suspect the IC and/or suspect shorts on the output signal paths. If the decoder inputs are bad as well, trace back toward the microprocessor. If your UUT has address latches or buffers, perform a similar test on them.

Watch for decoder ICs that are enabled only during reads or writes. Use the appropriate stimulus command *(read* or *write)* on these ICs.

Address Decode Circuit Example 4.11.3.

Figure 4-106 shows the address decode circuit (U8, U9, and U21) in the Demo/Trainer UUT. It selects the memory or I/O component being addressed. Some of the buffered address lines and bus controller lines are used to enable the following decoded address output lines (all have active low outputs):

Output	*Address Range Enabled*	*Circuit Enabled*
RAM0	00000-0FFFF	64K byte dynamic RAM
RAM1	10000-1FFFF	64K byte dynamic RAM
VRAM	20000-2FFFF	Video RAM
IPOLL	30000-3FFFF	Interrupt polling
SPARE1	40000-4FFFF	(decode complete signal)
SPARE2	50000-5FFFF	(decode complete signal)
ROM0	E0000-EFFFF	64K byte ROM, U29 and U30
ROM1	F0000-FFFFF	64K byte ROM, U27 and U28
VIDSLT	00000-01FFE	Video controller
I/OSLT	02000-03FFE	RS-232 port and the ASCII keyboard
PPISLT	04000-05FFE	Outputs to seven-segment displays and inputs from test switches S1 through S4

Keystroke Functional Test 4.11.4.

1. Use a 16-pin clip module on side A of I/O module 1 to test the decoded signals.

2. Use the SETUP MENU key with the commands below:

```
SETUP POD ENABLE ~READY OFF
SETUP POD REPORT FORCING SIGNALS OFF
```

3. Use the SYNC, I/O MOD, and STIM keys with the commands below for each of the following parts: U8, U9 and U21. The correct measurements for each pin are shown in the response table in Figure 4-106.

```
SYNC I/O MOD 1 TO POD DATA
ARM I/O MOD 1 FOR CAPTURE USING SYNC
RAMP ADDR 0 MASKED BY F0000
... (ADDR OPTION: MEMORY WORD)
RAMP ADDR 0 MASKED BY F000
... (ADDR OPTION: I/O BYTE)
SHOW I/O MOD 1 PIN <see response table> ...
... CAPTURED RESPONSES
```

NOTE

The SHOW command requires a clip module pin number rather than a part pin number. This requires you to translate part pin numbers to clip module pin numbers (see Appendix B of the Technical User's Manual). For your convenience, this translation has been done for you in this example, and the results are shown in the "I/O MOD PIN" column of the response table in Figure 4-106.

4. After completing the test, use the SETUP MENU key with the commands below to restore the settings for POD ENABLE and POD REPORT:

```
SETUP POD ENABLE READY ON
SETUP POD REPORT FORCING SIGNALS ON
```

(This page is intentionally blank.)

Keystroke Functional Test

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	I/O MOD U8 U9 U21

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
RAM0	U8-15	19	9 C A 8
RAM1	U8-14	18	6 B A D
VRAM	U8-13	17	9 0 2 F
IPOLL	U8-12	16	6 D E E
SPARE1	U8-11	15	9 3 0 E
SPARE2	U8-10	14	6 C 7 E
ROM0	U9-9	13	3 C 7 B
ROM1	U9-7	7	3 B 4 C
VIDSLT	U21-15	19	F 8 B E
I/OSLT	U21-14	18	0 9 2 A
PPISLT	U21-13	17	3 5 4 F

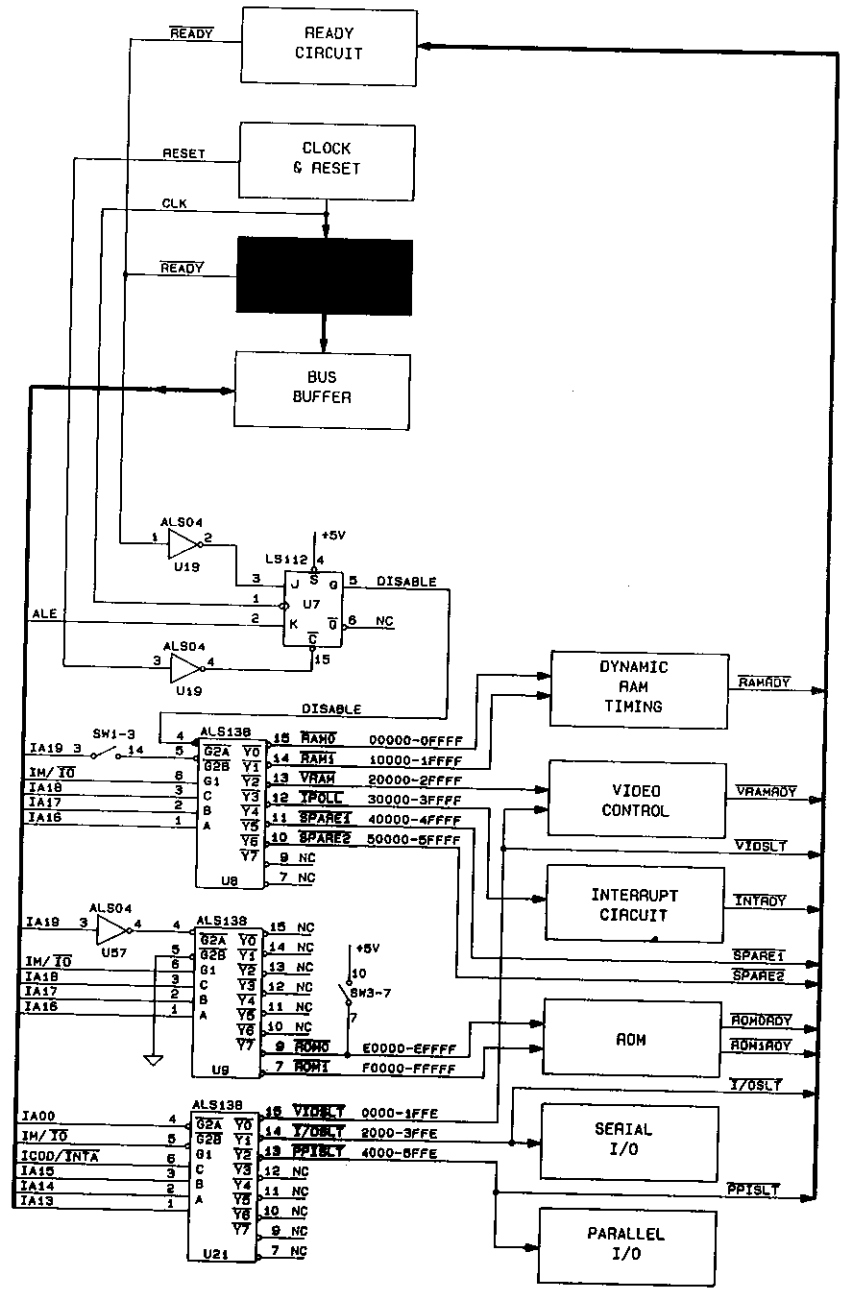


Figure 4-106: Address Decode Functional Test

Programmed Functional Test 4.11.5.

The *tst_decode* program is the programmed functional test for the Address Decode functional block. This program checks the three address decode ICs (U8, U9 and U21) using the *gfi test* command. If the *gfi test* command fails, the *abort_test* program is executed and GFI troubleshooting begins. (See the Bus Buffer functional block for a discussion of the *abort_test* program).

```
program tst_decode

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the DECODE functional block.                       !
!                                                                       !
! This program tests the DECODE functional block of the Demo/Trainer.   !
! The gfi test command and I/O module are used to clip over the decoders!
! and perform the test.                                                 !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    abort_test (ref-pin)                  If gfi has an accusation     !
!                                          display the accusation else  !
!                                          create a gfi hint for the    !
!                                          ref-pin and terminate the test!
!                                          program (GFI begins trouble- !
!                                          shooting).                   !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare
   global string  decode_checked = ""        ! Record this test was run
end declare

   if decode_checked <> "yes" then
      decode_checked = "yes"
      print "\nl\nlTESTING ADDRESS DECODE"

      podsetup 'enable ~ready' "off"
      podsetup 'report forcing' "off"

      if gfi test "U8-15" fails then abort_test("U8-15")
      if gfi test "U9-7" fails then abort_test("U9-7")
      if gfi test "U21-15" fails then abort_test("U21-15")

      print "ADDRESS DECODE TEST PASSES"
   end if
end program
```

Stimulus Programs and Responses 4.11.6.

Figure 4-107 is the stimulus program planning diagram for the Address Decode functional block. The *decode* stimulus program performs an access at each decoded address space. The *addr_out* stimulus program exercises the address lines. The *reset_low* stimulus program checks the reset signal when the test operator presses the Demo/Trainer UUT RESET pushbutton.

Stimulus Program Planning

PROGRAM: DECODE
PERFORMS AN ACCESS FOR EACH DECODED BLOCK
MEASUREMENT AT: U8-15,14,13,12,11,10 U9-9,7 U21-15,14,13 U7-5 U19-2,4

PROGRAM: RESET_LOW
PROMPTS THE OPERATOR TO PRESS THE RESET KEY AND THEN CHECKS FOR A LOW LEVEL
MEASUREMENT AT: U19-4

PROGRAM: ADDR_OUT
EXERCISES ALL ADDRESS LINES FROM THE MICROPROCESSOR
MEASUREMENT AT: U57-4

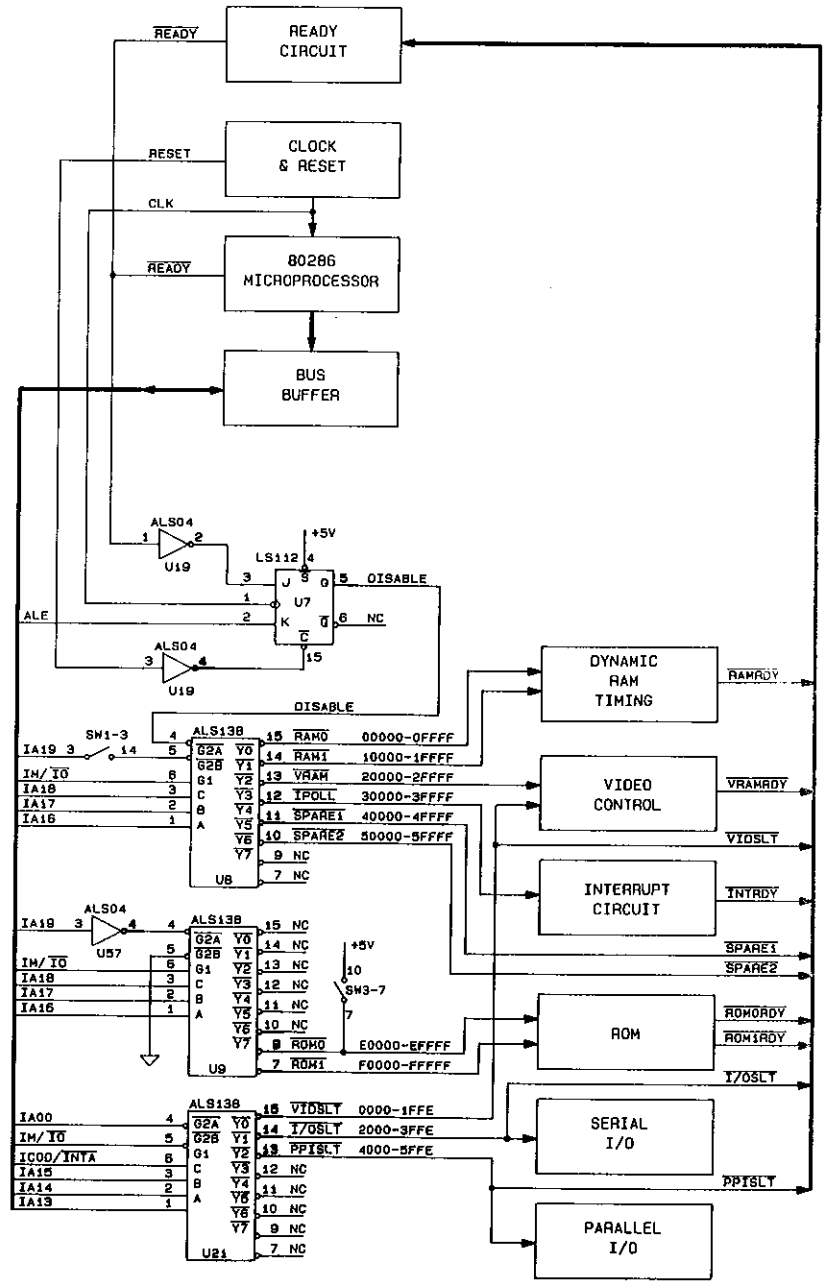


Figure 4-107: Address Decode Stimulus Program Planning

```
program decode

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM for address decoders U8, U9, and U21.                     !
!                                                                             !
! Stimulus programs and response files are used by GFI to backtrace          !
! from a failing node.  The stimulus program must create repeatable UUT      !
! activity and the response file contains the known-good responses for       !
! the outputs in the UUT that are stimulated by the stimulus program.        !
!                                                                             !
! This stimulus program is one of the programs which creates activity        !
! in the kernel area of the UUT.  These programs create activity with        !
! or without the ready circuit working properly.  Because of this, all       !
! the stimulus programs in the kernel area must disable the READY input      !
! to the pod, then perform the stimulus, then re-enable the READY input      !
! to the pod.  The 80286 microprocessor has a separate bus controller;       !
! for this reason, disabling ready and performing stimulus can get the       !
! bus controller out of synchronization with the pod.  Two fault             !
! handlers trap pod timeout conditions that indicate the bus controller      !
! is out of synchronization.  The recover() program is executed to           !
! resynchronize the bus controller and the pod.                              !
!                                                                             !
! TEST PROGRAMS CALLED:                                                       !
!    recover     ()                       The 80286 microprocessor has a!
!                                         bus controller that is totally!
!                                         separate from the pod.  In    !
!                                         some cases the pod can get out!
!                                         of sync with the bus control- !
!                                         ler.  The recover program     !
!                                         resynchronizes the pod and the!
!                                         bus controller.               !
!                                                                             !
! GRAPHICS PROGRAMS CALLED:                                                   !
!    (none)                                                                   !
!                                                                             !
! Local Variables Modified:                                                   !
!    devname                              Measurement device                  !
!                                                                             !
! Global Variables Modified:                                                  !
!    recover_times                        Reset to Zero                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
```

(continued on the next page)

Figure 4-108: Stimulus Program *(decode)*

```
handle pod_timeout_recovered
    recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine the measurement device.

    if (gfi control) = "yes" then
        devname = gfi device
    else
        devname = "/mod1"
    end if
    print "Stimulus Program DECODE"

! Set addressing mode and setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   io_byte = getspace space "i/o", size "byte"
   mem_word = getspace space "memory", size "word"
   reset device devname
   sync device devname, mode "pod"
   sync device "/pod", mode "data"
   old_cal = getoffset device devname
   setoffset device devname, offset (1000000 - 56)

! Present stimulus to UUT.

   arm device devname            ! Start response capture.
      setspace (mem_word)
      read addr 0                  ! RAM0
      read addr $10000             ! RAM1
      write addr $20000, data 0    ! VRAM (write only)
      read addr $30000             ! IPOLL
      read addr $40000             ! SPARE1
      read addr $50000             ! SPARE2
      read addr $E0000             ! ROM0
      read addr $F0000             ! ROM1
      setspace (io_byte)
      read addr 0                  ! VIDSLT
      read addr $2000              ! I/OSLT
      read addr $4000              ! PPISLT
   readout device devname        ! End response capture.
   setoffset device devname, offset old_cal

   podsetup 'enable ~ready' "on"
end program
```

Figure 4-108: Stimulus Program *(decode) - continued*

STIMULUS PROGRAM NAME: DECODE
DESCRIPTION: SIZE: 392 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Response Data: Clk LVL	Response Data: Counter Mode	Counter Range	Priority Pin
U8-15	I/O MODULE	03F9	1	0	TRANS		
U8-14	I/O MODULE	05F6	1	0	TRANS		
U8-13	I/O MODULE	06F1	1	0	TRANS		
U8-12	I/O MODULE	0772	1	0	TRANS		
U8-11	I/O MODULE	07B3	1	0	TRANS		
U8-10	I/O MODULE	07D3	1	0	TRANS		
U9-9	I/O MODULE	07E3	1	0	TRANS		
U9-7	I/O MODULE	07FB	1	0	TRANS		
U21-15	PROBE	07F7	1	0	TRANS		
U21-15	I/O MODULE	07F7	1	0	TRANS		
U21-14	PROBE	07F1	1	0	TRANS		
U21-14	I/O MODULE	07F1	1	0	TRANS		
U21-13	I/O MODULE	07F2	1	0	TRANS		
U7-5	I/O MODULE	0000	1	0	TRANS		
U19-2	I/O MODULE	0675	1	0	TRANS		
U19-4	I/O MODULE	07F3	1		TRANS		
U45-3	I/O MODULE	07FB	1	0	TRANS		
U45-6	I/O MODULE	07E3	1	0	TRANS		
U5-11	I/O MODULE	07F3	1		TRANS		
U4-3	I/O MODULE	07F3	1		TRANS		
U57-2	I/O MODULE	0637	1	0	TRANS		
U57-6	I/O MODULE	0081	1	0	TRANS		

Figure 4-109: Response File *(decode)*

Summary of Complete Solution for Address Decode 4.11.7.

The entire set of programs and files needed to test and GFI troubleshoot the Address Decode functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Address Decode)

Programs (PROGRAM):

TST_DECODE	Functional Test	Section 4.11.5
DECODE	Stimulus Program	Figure 4-108
ADDR_OUT	Stimulus Program	Figure 4-4
RESET_LOW	Stimulus Program	Figure 4-115

Stimulus Program Responses (RESPONSE):

DECODE	Figure 4-109
ADDR_OUT	Figure 4-5
RESET_LOW	Figure 4-114

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

(This page is intentionally blank.)

CLOCK AND RESET FUNCTIONAL BLOCK 4.12.

Introduction to Clock and Reset Circuits 4.12.1.

Microprocessor-system clock circuits may generate single periodic digital signals or multiple signals representing different phases of a single time base. Both types of clocks may be present in a UUT. Clock circuits typically include circuitry for buffering and/or dividing clock sources.

Reset circuits range in complexity from simple resistor-capacitor networks to several IC's. Often a single switch, IC, gate, or monostable multivibrator serves as the reset circuit. Some UUTs have watchdog timers which automatically reset the UUT if the microprocessor gets lost in a program.

Considerations for Testing and Troubleshooting 4.12.2.

Clocks

When clocks circuits fail, most other functional blocks will also fail. Clock problems are usually associated with only a few components. Here are some guidelines:

- *Open or stuck nodes* on the crystal oscillator. Manufacturing defects or failed components may cause stuck or open lines on ICs used as oscillators.
- *DC or capacitive loading* on the outputs of the oscillator. A stuck or tied line may load the oscillator output so that it cannot generate a signal.
- *Failed counter or flip-flop* deriving lower frequency signals from the master clock. Pullup or pulldown resistors establishing static logic levels on unused counter or flip-flop inputs may be short or open.

- *Failed clock-generator IC.* Clock generator ICs may fail due to manufacturing defects or shorted or tied inputs.

Frequency measurements with the probe or I/O module are a good way to trace clock-related problems. For measurements above 10 MHz, use the probe; measurements below that frequency can be made with the I/O module.

The Demo/Trainer UUT stimulus program called *frequency,* in Section 4.12.6, shows how to program the I/O module to measure frequency. The frequency of the clock is measured three times during a 9100A/9105A LEARN operation on a known-good UUT, when the response file is created. If the value of the clock is stable, a single decimal value is recorded. If the value of the clock is unstable, the highest and lowest values are recorded. With frequency or transition counts, the min-max range must be large enough to account for variations between UUTs and variations due to environmental factors, such as temperature and humidity. To establish the range, first learn the response from a known-good UUT, then adjust the range for appropriate tolerance factors.

Some clock-related problems, such as injected noise, marginal signals, or asymmetrical phases, are hard to detect with digital test equipment. The probe, which operates at up to 40 MHz, is very useful for these problems. Asynchronous level history measurements with the probe can detect marginal signal levels and noise. If, after measurements with the probe, the UUT still exhibits erratic clock behavior, check the quality of the clock signal with a high-bandwidth oscilloscope.

Reset

Asynchronous level history is a useful measurement technique with which to verify the operation of a reset circuit.

Several 9100A/9105A devices are useful in detecting reset faults. The probe can be used to verify static logic levels on circuit nodes. The I/O module can be used to overdrive the Reset input to verify operation. Since most Reset lines connect

to the microprocessor, the pod can sense whether this line is active. In setting up test fixturing, it is helpful to connect the Reset line to a test point or test connector attached to an I/O module line. This allows the test program to automatically reset the UUT at the start of a test sequence.

Verify operation of the Reset line in both states. The Demo/Trainer UUT stimulus programs called *reset_low* and *reset_high*, in Section 4.12.6, show how the probe and I/O module can be used to troubleshoot reset circuits.

For reset circuits that use a switch or pushbutton, the operator must usually be involved. A prompt to the operator can be displayed, asking that the switch be pressed during certain modes of the test while measurements are performed.

Clock and Reset Example 4.12.3.

The clock source in the Demo/Trainer UUT is a 31.9399 MHz oscillator (U18). This frequency is divided by two and by four. The 8 MHz signal is used by the 82284 clock generator (U1) to generate the microprocessor clock signals. The 31.9399 MHz signal is also used in the Video Ready generation circuit.

The 15.9799 MHz signal is used as the clock source for the video circuit. The Reset signal is controlled by the RESET pushbutton switch. Pressing this switch causes an active Ready signal to be generated.

Keystroke Functional Test 4.12.4.

Part A:

Measure frequency of clock signals with the probe, using the PROBE and SOFT KEYS key with the command below:

```
FREQ AT PROBE =
```

The pins to be probed and the correct measurements at each pin are shown in the response table in Figure 4-110.

Part B:

Operate the RESET switch and measure the level of U1-12 with the probe, using the PROBE and SOFT KEYS key with the command below:

```
INPUT PROBE LEVEL =
```

The pins to be probed and the correct measurements at each pin are shown in the response table in Figure 4-111.

(This page is intentionally blank.)

Keystroke Functional Test (Part A)

CONNECTION TABLE

	MEASUREMENT
(NONE)	PROBE U1 U25

RESPONSE TABLE

SIGNAL	PART PIN	FREQUENCY	
		MINIMUM	MAXIMUM
CCLK	U1-10	7.995 MHZ	8.000 MHZ
PCLK	U1-13	3.997 MHZ	4.000 MHZ
16 MHZ	U25-9	15.99 MHZ	16.00 MHZ
32 MHZ	U25-13	31.98 MHZ	32.00 MHZ

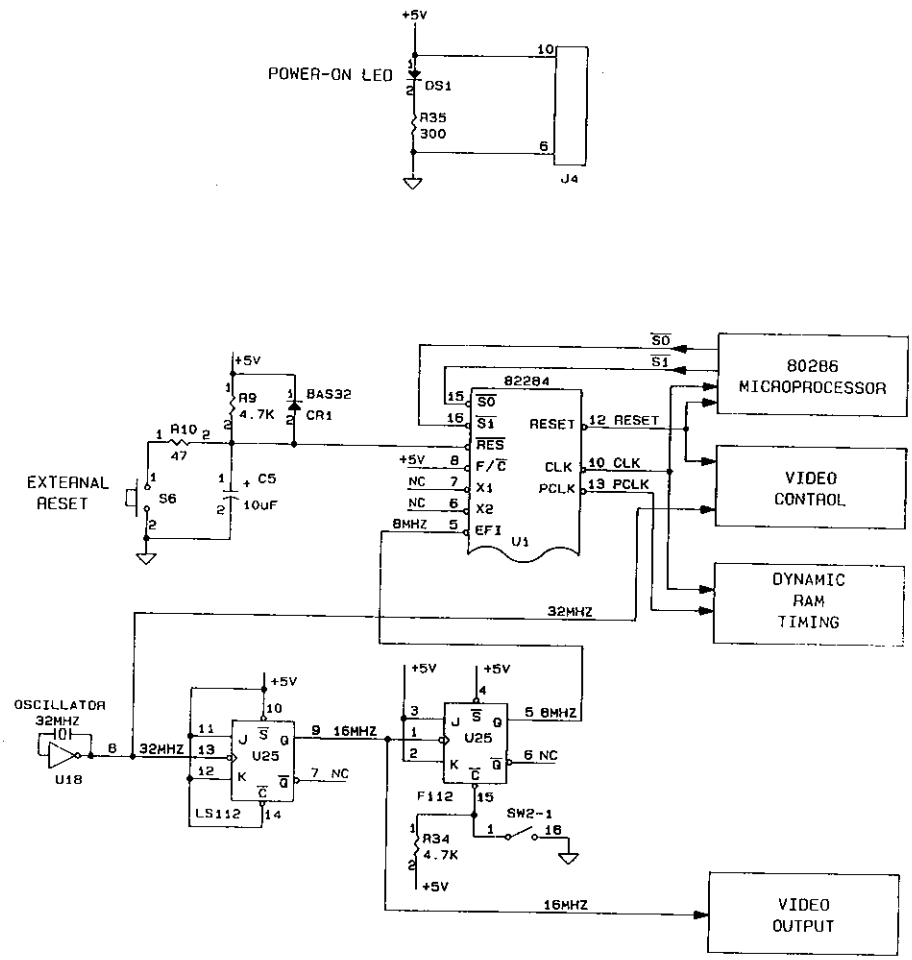


Figure 4-110: Clock and Reset Functional Test (Part A)

Keystroke Functional Test (Part B)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] S6	PROBE U1-12

STIMULUS AND RESPONSE TABLE

[redacted]	PART PIN	LEVEL
	U1-12	LOW
	U1-12	HIGH

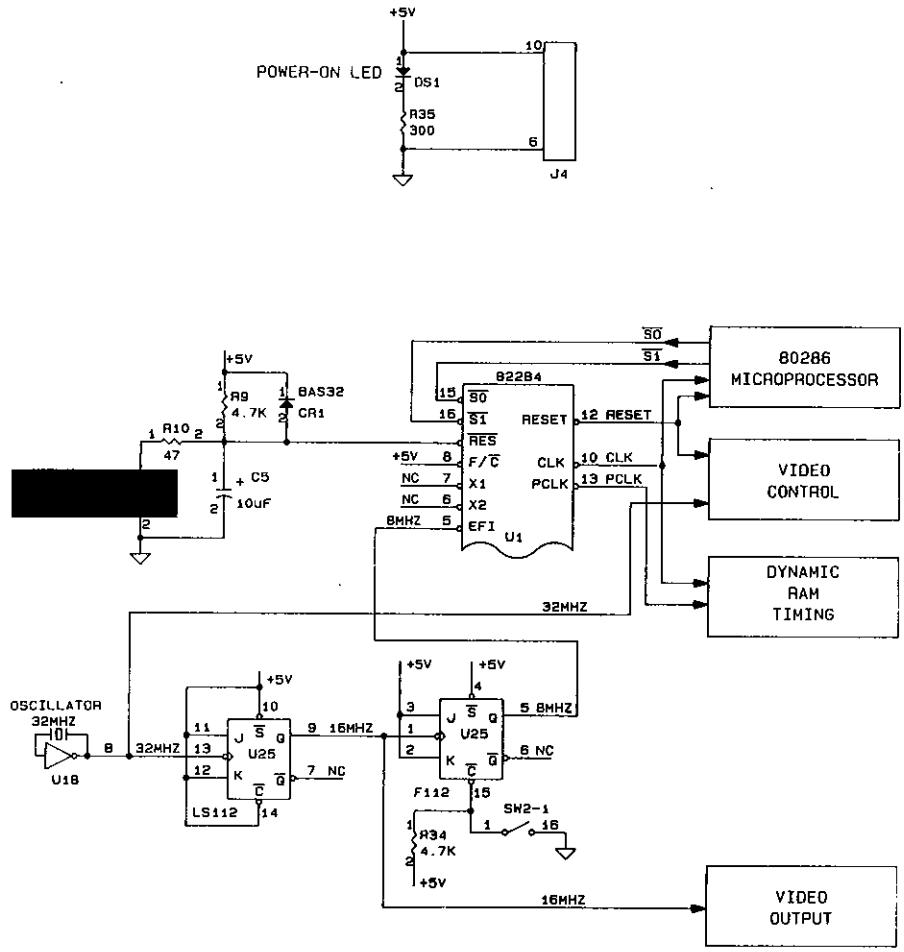


Figure 4-111: Clock and Reset Functional Test (Part B)

Programmed Functional Test 4.12.5.

The *tst_clock* program is the programmed functional test for the Clock and Reset functional block. U1 is a signal conditioning IC for the Clock, Reset, and Ready signals, however the *tst_clock* program tests only the Clock and Reset portion of the chip.

The *tst_clock* program uses the *gfi status* command to determine if U1 has previously been tested using *gfi test*. If U1 has not been tested, a *gfi test* of U1 is performed. The *gfi status* command is then used to determine if the Clock and Reset outputs of U1 failed. If the outputs failed, the *abort_test* program is executed and GFI troubleshooting is started. (See the Bus Buffer functional block for a discussion of *abort_test*).

```
program tst_clock

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the CLOCK and RESET functional block.                    !
!                                                                            !
! This program tests the CLOCK and RESET functional block of the             !
! Demo/Trainer.  The gfi test command, I/O module and PROBE are used to      !
! perform the test.                                                          !
!                                                                            !
! TEST PROGRAMS CALLED:                                                      !
!    abort_test (ref-pin)                     If gfi has an accusation       !
!                                             display the accusation else    !
!                                             create a gfi hint for the      !
!                                             ref-pin and terminate the test!
!                                             program (GFI begins trouble-   !
!                                             shooting).                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   print "\nlTESTING CLOCK & RESET Circuit"

   if (gfi status "U1-10") = "untested" then
      gfi test "U1-10"
   end if
   if (gfi status "U1-12") = "bad" then abort_test("U1-12")
   if (gfi status "U1-10") = "bad" then abort_test("U1-10")
   if (gfi status "U1-13") = "bad" then abort_test("U1-13")
   if gfi test "U25-9" fails then abort_test("U25-9")

   print "CLOCK & RESET TEST PASSES"
end program
```

Stimulus Programs and Responses 4.12.6.

Figure 4-112 is the stimulus program planning diagram for the Clock and Reset functional block. *frequency* is a general-purpose stimulus program used to measure the frequencies of various outputs around the Demo/Trainer UUT. *reset_high* checks for a high-level Reset signal and *reset_low* checks for a low-level Reset signal.

Stimulus Program Planning

PROGRAM: FREQUENCY
MEASURES FREQUENCY
MEASUREMENT AT: U25-9,5 U1-10,13

PROGRAM: RESET_LOW
PROMPTS THE OPERATOR TO PRESS THE RESET KEY AND THEN CHECKS FOR A LOW LEVEL
MEASUREMENT AT: R10-1 R9-2

PROGRAM: RESET_HIGH
PROMPTS THE OPERATOR TO PRESS THE RESET KEY AND THEN CHECKS FOR A HIGH LEVEL
MEASUREMENT AT: U1-12

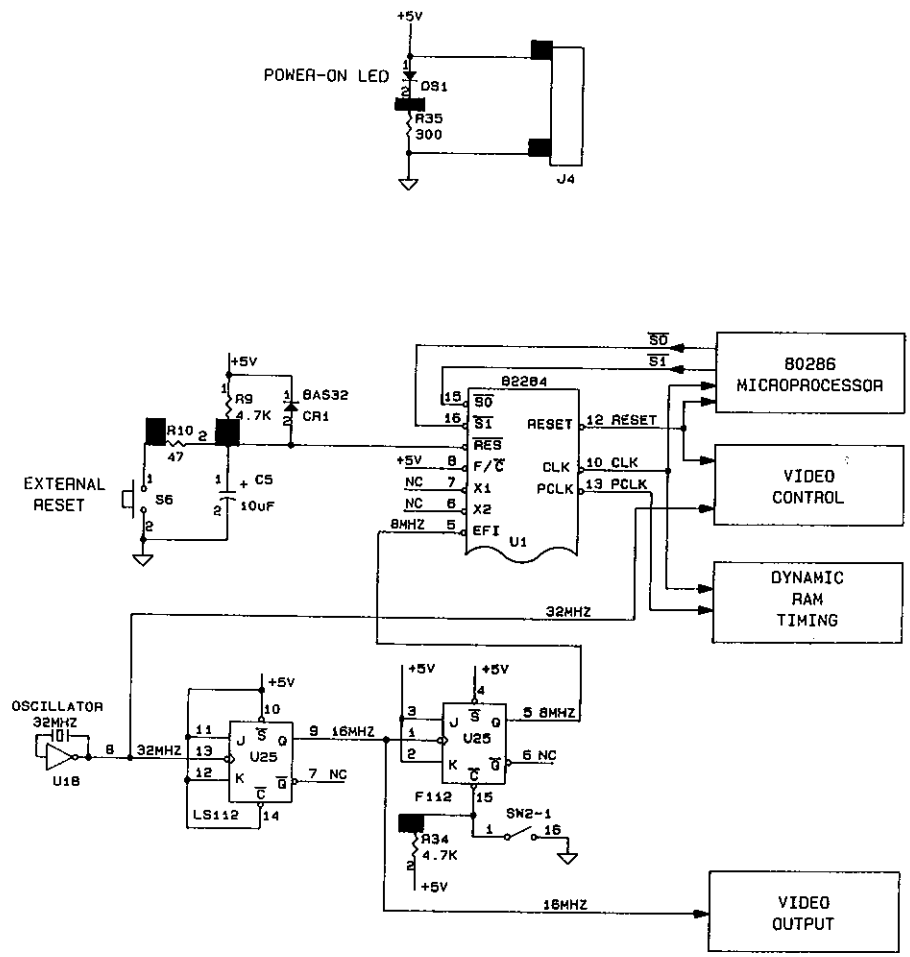


Figure 4-112: Clock and Reset Stimulus Program Planning

```
program reset_high

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM characterizes the reset signal when high is active.  !
!                                                                       !
! Stimulus programs and response files are used by GFI to back-trace    !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!     (none)                                                            !
!                                                                       !
! GRAPHICS PROGRAMS CALLED:                                             !
!     (none)                                                            !
!                                                                       !
! Local Constants Modified:                                             !
!    CARRAGE_RETURN                  Matches a carrage return input.    !
!    TRUE                            Value that is considered active TRUE!
!                                                                       !
! Local Variables Modified:                                             !
!    input_str                       Input from keypad                  !
!    state                           Level returned from measurement    !
!    pinnum                          The pin number used by level command!
!    finished                        State of loop looking for condition !
!    devname                         Measurement device                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare string CARRAGE_RETURN = ""
   declare numeric TRUE = 1
   declare string input_str
   declare numeric state = 0
   declare numeric pinnum = 1
   finished = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the testing device.

   if (gfi control) = "yes" then
      devname = gfi device
      measure_ref = gfi ref
      if measure_ref = "U1" then pinnum = 12
      if measure_ref = "U11" then pinnum = 38
```

(continued on the next page)

Figure 4-113: Stimulus Program *(reset_high)*

```
      if measure_ref = "U13" then pinnum = 11
      if measure_ref = "U31" then pinnum = 35
      if measure_ref = "U19" then pinnum = 3
      if measure_ref = "U7" then pinnum = 15
   else
      devname = clip ref "U1"
      measure_ref = "U1"
   end if
   print "Stimulus Program RESET_HIGH"

! Setup measurement device and prompt operator.

   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   reset device devname
   sync device devname, mode "int"
   podsetup 'report forcing' "off"
   tlup = open device "/term1", as "update"
   print on tlup ,"\07WHILE MEASURING, Press \1B[7mDemo UUT RESET\1B[0m key."
   print on tlup ,"Press 9100 ENTER key if test is stuck."

! Wait for a TRUE.  Leave program if <ENTER> key is pressed.

   loop until state = TRUE
      arm device devname \ readout device devname
      if devname = "/probe" then
         state = level device devname, type "async"
      else
         state = level device measure_ref, pin pinnum, type "async"
      end if
      if (poll channel tlup, event "input") = 1 then
         input on tlup ,input_str
         if input_str = CARRAGE_RETURN then return
      end if
   end loop

! Start response capture.  End when POD detects reset.

   arm device devname
      strobeclock device devname
      loop until finished = 1
         x = readstatus()
         if (x and $10) = $10 then
            strobeclock device devname
            finished = 1
         end if
         if (poll channel tlup, event "input") = 1 then
            input on tlup ,input_str
            if input_str = CARRAGE_RETURN then return
         end if
      end loop
   readout device devname
   print "\n1\n1"

end program
```

Figure 4-113: Stimulus Program *(reset_high) - continued*

```
STIMULUS PROGRAM NAME: RESET_HIGH
DESCRIPTION:                                        SIZE:          78 BYTES

                  ------------------- Response Data --------------------
   Node            Learned                Async Clk Counter                    Priority
Signal Src          With          SIG       LVL  LVL  Mode     Counter Range     Pin

U1-12               PROBE         0001      1  0      TRANS
U1-12               I/O MODULE    0001      1  0      TRANS
```

Figure 4-114: Response File *(reset_high)*

```
program reset_low

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM characterizes the reset signal when low is active.       !
!                                                                           !
! Stimulus programs and response files are used by GFI to backtrace        !
! from a failing node.  The stimulus program must create repeatable UUT    !
! activity and the response file contains the known-good responses for     !
! the outputs in the UUT that are stimulated by the stimulus program.       !
!                                                                           !
! TEST PROGRAMS CALLED:                                                     !
!                                                                           !
!    check_meas (device, start, stop, clock, enable)                        !
!                                         Checks to see if the measure-    !
!                                         ment is complete using the       !
!                                         TL/1 checkstatus command.  If    !
!                                         the measurement times out then!
!                                         redisplay connect locations.     !
!                                                                           !
! GRAPHICS PROGRAMS CALLED:                                                 !
!    (none)                                                                 !
!                                                                           !
! Local Constants Modified:                                                 !
!    CARRAGE_RETURN                  Matches a carrage return input.        !
!    TRUE                            Value that is considered active true!
!                                                                           !
! Local Variables Modified:                                                 !
!    input_str                       Input from keypad                      !
!    state                           Level returned from measurement        !
!    pinnum                          The pin number used by level command!
!    finished                        State of loop looking for condition !
!    devname                         Measurement device                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare string CARRAGE_RETURN = ""
   declare string input_str
   declare numeric state = 0
   declare numeric TRUE = 4
   declare numeric pinnum = 1
   finished = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the testing device.

   if (gfi control) = "yes" then
      devname = gfi device
      measure_ref = gfi ref
```

(continued on the next page)

Figure 4-115: Stimulus Program *(reset_low)*

```
      if measure_ref = "U1" then pinnum = 11
      if measure_ref = "U13" then pinnum = 13
      if measure_ref = "U19" then pinnum = 4
      if measure_ref = "U7" then pinnum = 15
   else
      devname = clip ref "U1"
      measure_ref = "U1"
   end if
   print "Stimulus Program RESET_LOW"

! Setup measurement device and prompt operator.

   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   reset device devname
   sync device devname, mode "int"
   podsetup 'report forcing' "off"
   tlup = open device "/term1", as "update"
   print on tlup ,"\07WHILE MEASURING, Press \1B[7mDemo UUT RESET\1B[0m key."
   print on tlup ,"Press 9100 ENTER key if test is stuck."

! Wait for a TRUE.  Leave program if <ENTER> key is pressed.

   loop until state = TRUE
      arm device devname \ readout device devname
      if devname = "/probe" then
         state = level device devname, type "async"
      else
         state = level device measure_ref, pin pinnum, type "async"
      end if
      if (poll channel tlup, event "input") = 1 then
         input on tlup ,input_str
         if input_str = CARRAGE_RETURN then return
      end if
   end loop

! Start response capture.  End when POD detects reset.

   arm device devname
      strobeclock device devname
      loop until finished = 1
         x = readstatus()
         if (x and $10) = $10 then
            strobeclock device devname
            finished = 1
         end if
         if (poll channel tlup, event "input") = 1 then
            input on tlup ,input_str
            if input_str = CARRAGE_RETURN then return
         end if
      end loop
   readout device devname
   print "\n1\n1"

end program
```

Figure 4-115: Stimulus Program *(reset_low) - continued*

```
STIMULUS PROGRAM NAME: RESET_LOW
DESCRIPTION:                                          SIZE:          146 BYTES

                    ------------------- Response Data --------------------
   Node      Learned                    Async Clk Counter                    Priority
Signal Src    With        SIG            LVL  LVL  Mode     Counter Range     Pin

U13-10       PROBE       0002           1  0        TRANS
U13-10       I/O MODULE  0002           1  0        TRANS
U19-4        I/O MODULE  0002           1  0        TRANS
R10-1        PROBE       0002              1  0 TRANS
R9-2         PROBE       0002              1  0 TRANS
R9-2         I/O MODULE  0002              1  0 TRANS
```

Figure 4-116: Response File *(reset_low)*

```
program frequency

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to measure frequencies.                               !
!                                                                        !
! Stimulus programs and response files are used by GFI to backtrace      !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
! This is a general purpose routine that can be used to characterize    !
! any free-running system clock, dot clock, etc...                       !
! When measuring frequency no stimulus is normally applied because the  !
! signal begins running at power on.                                     !
!                                                                        !
! TEST PROGRAMS CALLED:                                                  !
!    (none)                                                              !
!                                                                        !
! GRAPHICS PROGRAMS CALLED:                                              !
!    (none)                                                              !
!                                                                        !
! Local Variables Modified:                                              !
!    devname                              Measurement device             !
!                                                                        !
! Global Variables Modified:                                             !
!    (none)                                                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  FAULT HANDLERS:                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
handle pod_timeout_no_clk
end handle
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Main part of STIMULUS PROGRAM                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program FREQUENCY"

! Set addressing mode and setup measurement device.

   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   reset device devname
   counter device devname, mode "freq"

! No stimulus is applied; response is frequency.

   arm device devname        ! Start response capture.
   readout device devname    ! End response capture.
end program
```

Figure 4-117: Stimulus Program *(frequency)*

STIMULUS PROGRAM NAME: FREQUENCY
DESCRIPTION: SIZE: 370 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U1-10	PROBE		1	0	FREQ	7585000-8383000	U25-5
U1-10	I/O MODULE		1	0	FREQ	7585000-8383000	
U1-13	I/O MODULE		1	0	FREQ	3792000-4191000	U25-5
U25-5	PROBE		1	0	FREQ	7585000-8383000	
U25-5	I/O MODULE		1	0	FREQ	7585000-8383000	
U25-9	PROBE		1	0	FREQ	15170000-16760000	
U42-3	I/O MODULE		1	0	FREQ	379200-419100	
U42-7	I/O MODULE		1	0	FREQ	758500-838300	
U43-11	I/O MODULE		1	0	FREQ	63200-69800	
U56-12	PROBE		1	0	FREQ	63200-69800	
U56-12	I/O MODULE		1	0	FREQ	63200-69800	
U13-2	PROBE		1	0	FREQ	7585000-8383000	
U13-2	I/O MODULE		1	0	FREQ	7585000-8383000	
Y1-1	PROBE		1	0	FREQ	3670000-3700000	

Figure 4-118: Response File *(frequency)*

Summary of Complete Solution for Clock and Reset 4.12.7.

The entire set of programs and files needed to test and GFI troubleshoot the Clock and Reset functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Clock and Reset)

Programs (PROGRAM):

TST_CLOCK	Functional Test	Section 4.12.5
FREQUENCY	Stimulus Program	Figure 4-117
RESET_HIGH	Stimulus Program	Figure 4-113
RESET_LOW	Stimulus Program	Figure 4-115
LEVELS	Stimulus Program	Figure 4-92

Stimulus Program Responses (RESPONSE):

FREQUENCY	Figure 4-118
RESET_HIGH	Figure 4-114
RESET_LOW	Figure 4-116
LEVELS	Figure 4-93

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

INTERRUPT CIRCUIT FUNCTIONAL BLOCK 4.13.

Introduction to Interrupt Circuits 4.13.1.

Microprocessor-system interrupt circuits collect and prioritize the interrupt output of each circuit that has an interrupt-request output. These outputs come from circuits such as peripheral devices (keyboards, disk controllers, modems, printers) and dynamic RAM controllers. If there are enough interrupt signals, the system may use an interrupt controller to prioritize interrupts.

In some systems, the microprocessor can read a pointer to a branch address (called an "interrupt vector") from the microprocessor's external bus. These systems may have circuitry to generate the interrupt vector when the appropriate interrupt signal is asserted. Quite often, the vector-generation and interrupt-controller circuits are the same.

Figure 4-119 shows a typical interrupt circuit for a microprocessor system.

Considerations for Testing and Troubleshooting 4.13.2.

The Interrupt Circuit is part of a feedback loop. Address and data buses go out from the microprocessor to the various components in the UUT and interrupt lines come back from those components, through the Interrupt Circuit, to the microprocessor.

The pod can break this feedback loop by selectively ignoring the interrupt line to the pod. Particularly during troubleshooting, the interrupt line must be ignored so the 9100A/9105A is not interrupted while testing the interrupt circuitry.

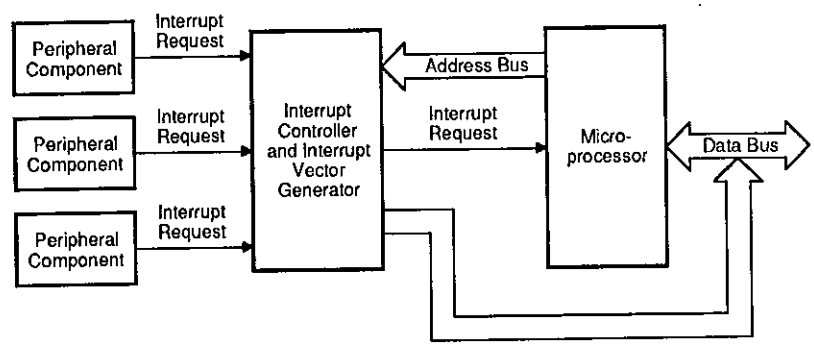


Figure 4-119: Typical Interrupt Circuit

The Interrupt Circuit can be tested by the following procedure:

1. Read or write to each component that can generate an interrupt so that an interrupt is generated.

2. After each interrupt is generated, check to see that the pod has detected the interrupt. If all interrupts are detected by the pod, the interrupt circuit is good.

If the microprocessor on your UUT has the ability to fetch an interrupt vector from its external bus, test the circuit that generates that vector by reading or writing to a component and thereby forcing that component to generate an interrupt. The interrupt vector should be the same address as the read or write address used to generate the interrupt.

Some pods (e.g. 8086, 8088, 80186, 80188, 80286, 68000) can read interrupt vectors. The '86-family and '88-family pods, for example, can read vectors automatically in response to an interrupt input from the pod to the UUT, or by command from the operator (TL/1 programs that perform these functions are accessed with the POD key on the operator's keypad).

The availability of these automatic interrupt testing functions greatly eases the test procedures. With these functions, the procedure for testing interrupt vector generation circuits might work like this:

1. Configure the pod to capture an interrupt vector (this is usually called an "interrupt acknowledge cycle").

2. Write the interrupt vector to the interrupt controller or vector generator.

3. Perform some operation that causes the interrupt controller to interrupt the pod and place a vector on the UUT's bus. This operation may simply mean overdriving an input to the interrupt controller.

Troubleshooting the interrupt circuitry is accomplished by performing a procedure that causes each circuit with an interrupt request output to activate that output. Then signatures are recorded for all the nodes in the Interrupt Circuit. The steps to perform this are as follows:

1. Generate an interrupt on each interrupt request line that feeds into the interrupt circuit by performing the appropriate reads and writes.

2. Measure the signatures for each node in the Interrupt Circuit and compare to known-good signatures.

3. If an incorrect signature is found, follow that signal back towards its source.

You may need to disable the reporting of active interrupts by the pod when troubleshooting this circuit. If reporting is allowed and the interrupt is asserted, you may be unnecessarily bothered with "active interrupt" messages when the pod is used in stimulus operations. Section 4.15.2, "Forcing Lines", in this manual describes how to disable reporting of active interrupts.

Interrupt Circuit Example 4.13.3.

Figure 4-120 shows the Interrupt Circuit for the Demo/Trainer UUT. This circuit uses two interrupts. The first, I/OINT, is configurable to be active when a character is transmitted or received through the serial port. The second, TIMER, is configurable to be active when the timer in the DUART IC (in the Serial I/O functional block) times out or when the output port toggles the bit in the output register connected to the TIMER output line.

Keystroke Functional Test 4.13.4.

1. Use the SETUP MENU, EXEC, and READ keys with the commands below to disable interrupt trapping and to

initialize the Serial I/O functional block:

```
SETUP POD REPORT INTR ACTIVE OFF
EXECUTE RS232_INIT
READ ADDR 2016 =
... (ADDR OPTION: I/O BYTE)
```

2. Use the READ key with the commands below to check the status of interrupts in the UUT:

```
READ STATUS OF MICRO =
     (Should be C0 with no interrupts)
READ ADDR 30000 =
... (ADDR OPTION: MEMORY WORD)
     (Should be 27 with no interrupts)
```

3. Use the WRITE and READ keys with the following commands to force an interrupt on TIMER (by setting output OP3 low) and to check that the interrupt occurs:

```
WRITE DATA 0 TO ADDR 201A
... (ADDR OPTION: I/O BYTE)
WRITE DATA 8 TO ADDR 201C
... (ADDR OPTION: I/O BYTE)
READ STATUS OF MICRO =
     (Should be C8 with an interrupt)
READ ADDR 30000 =
... (ADDR OPTION: MEMORY WORD)
     (Should be 25 with a TIMER interrupt)
WRITE DATA 8 TO ADDR 201E
... (ADDR OPTION: I/O BYTE)
```

4. Use the WRITE and READ keys with the following commands to force an interrupt on I/OINT (by causing an interrupt from RS232) and to check that this interrupt occurs:

```
WRITE DATA 10 TO ADDR 200A
... (ADDR OPTION: I/O BYTE)
WRITE DATA 41 TO ADDR 2016
... (ADDR OPTION: I/O BYTE)
```

```
READ STATUS OF MICRO =
     (Should be C8 with an interrupt)
READ ADDR 30000 =
     (Should be 22 with the I/OINT interrupt)
... (ADDR OPTION: MEMORY WORD)
WRITE DATA 0 TO ADDR 200A
... (ADDR OPTION: I/O BYTE)
```

5. Re-enable interrupt trapping by using the SETUP MENU key to enter the following command:

```
SETUP POD REPORT INTR ACTIVE ON
```

(This page is intentionally blank.)

Keystroke Functional Test

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET	POD TEST ACCESS SOCKET

STIMULUS AND MEASUREMENT TABLE

[redacted]	MEASUREMENT	
	STATUS	ADDRESS 30000
	C0 C8 C8	27 25 22

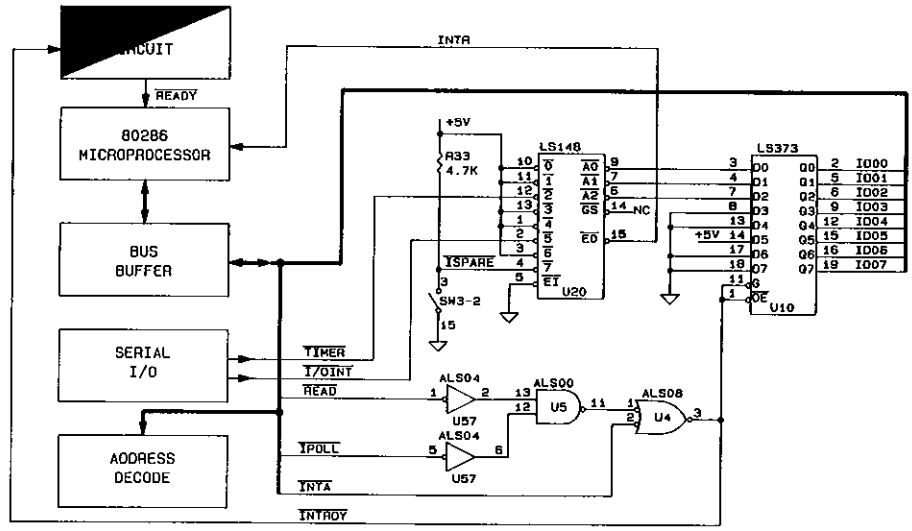


Figure 4-120: Interrupt Circuit Functional Test

Programmed Functional Test 4.13.5.

The *tst_intrpt* program is the programmed functional test for the Interrupt Circuit functional block. This program checks the interrupt poll register using the *gfi test* command. If the *gfi test* command fails, the *abort_test* program is executed and GFI troubleshooting begins. (See the Bus Buffer functional block for a discussion of the *abort_test* program).

```
program tst_intrpt

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the INTERRUPT functional block.                     !
!                                                                       !
! This program tests the INTERRUPT functional block of the Demo/Trainer.!
! The gfi test command and I/O module are used to perform the test.     !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    abort_test (ref-pin)                   If gfi has an accusation    !
!                                           display the accusation else !
!                                           create a gfi hint for the   !
!                                           ref-pin and terminate the test!
!                                           program (GFI begins trouble- !
!                                           shooting).                  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   print "\nlTESTING INTERRUPT Circuit"

   podsetup 'report intr' "off"
   if gfi test "U10-1" fails then abort_test("U10-1")

   print "INTERRUPT TEST PASSES"
end program
```

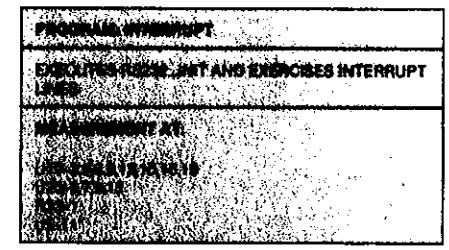
Stimulus Programs and Responses 4.13.6.

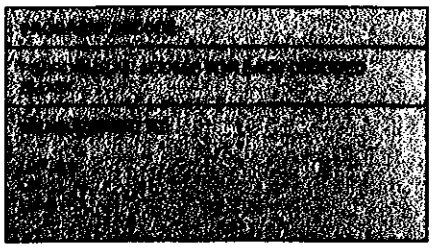
Figure 4-121 is the stimulus program planning diagram for the Interrupt Circuit functional block. The *decode* stimulus program performs an access at each decoded address space. The *ttl_lvl* stimulus program transmits a character out the serial port and measures signals using TTL threshold levels. The *interrupt* stimulus program generates interrupts in the Serial I/O circuit and measures interrupt lines.

(This page is intentionally blank.)

Stimulus Program Planning

PROGRAM: CTRL_OUT3
PERFORMS AN ACCESS FOR EACH DECODED BLOCK
MEASUREMENT AT: U4-3





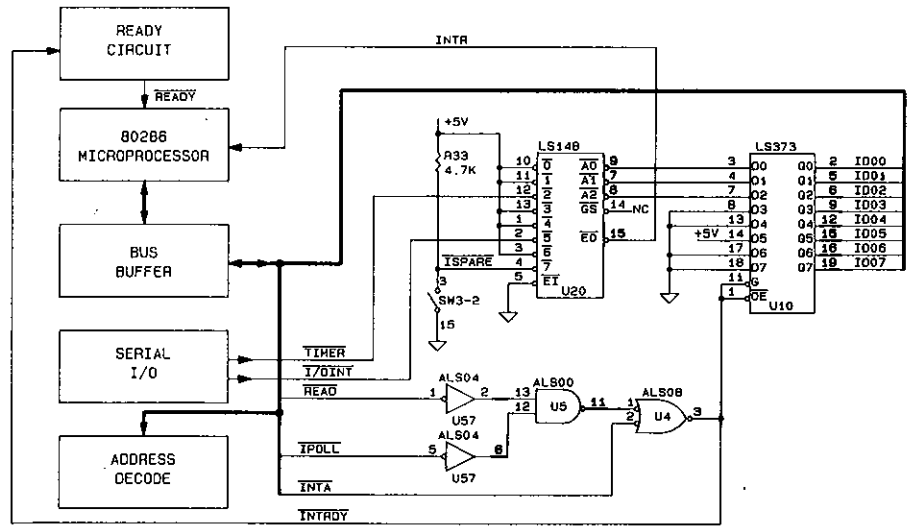


Figure 4-121: Interrupt Circuit Stimulus Program Planning

```
program interrupt

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to exercise the interrupt circuitry.                      !
!                                                                            !
! Stimulus programs and response files are used by GFI to backtrace         !
! from a failing node.  The stimulus program must create repeatable UUT     !
! activity and the response file contains the known-good responses for      !
! the outputs in the UUT that are stimulated by the stimulus program.       !
!                                                                            !
! This stimulus program sets the DUART to cause an interrupt when data      !
! is written to the transmit register.  Immediately after the write to      !
! the register the interrupt vector is read from the bus (read @ 30000).!
!                                                                            !
! TEST PROGRAMS CALLED:                                                      !
!    (none)                                                                  !
!                                                                            !
! GRAPHICS PROGRAMS CALLED:                                                  !
!    rs232_init()                       This is the initalization for       !
!                                       the DUART which contains a          !
!                                       timer used for interrupts.          !
!                                                                            !
! Local Variables Modified:                                                  !
!    devname                            Measurement device                  !
!                                                                            !
! Global Variables Modified:                                                 !
!    (none)                                                                  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Main part of STIMULUS PROGRAM                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      devname = gfi device
   else
      devname = "/mod1"
   end if
   print "Stimulus Program INTERRUPT"

! Set addressing mode and setup measurement device.

   reset device devname
   execute rs232_init()
   write addr $200A, data $10  ! Set interrupt on tranmit - no loopback
   setspace space (getspace space "i/o", size "byte")
   sync device devname, mode "pod"
   sync device "/pod", mode "data"
   threshold device "/probe", level "ttl"
```

(continued on the next page)

Figure 4-122: Stimulus Program *(interrupt)*

```
! Present stimulus to UUT.

   arm device devname                ! Start response capture.
      write addr $2016, data $55     !    Txd port B
      setspace space (getspace space "memory", size "word")
      read addr $30000               ! read the interrupt vector onto the bus.
      setspace space (getspace space "i/o", size "byte")
      write addr $2016, data  $D     !    Txd port B
      setspace space (getspace space "memory", size "word")
      read addr $30000
      setspace space (getspace space "i/o", size "byte")
      write addr $201C, data $FF
      setspace space (getspace space "memory", size "word")
      read addr $30000
      setspace space (getspace space "i/o", size "byte")
      write addr $201E, data $FF     !    Pulse timer interrupt.
      setspace space (getspace space "memory", size "word")
      read addr $30000
   readout device devname            ! End response capture.

end program
```

Figure 4-122: Stimulus Program *(interrupt) - continued*

STIMULUS PROGRAM NAME: INTERRUPT
DESCRIPTION: SIZE: 660 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U10-6	PROBE	00AB	1 0	1 0	TRANS	4	
U10-6	I/O MODULE	00AB	1 0	1 0	TRANS	4	
U10-2	PROBE	00AB	1 0	1 0	TRANS	4	
U10-2	I/O MODULE	00AB	1 0	1 0	TRANS	4	
U10-5	PROBE	005F	1 0	1 0	TRANS	2	
U10-5	I/O MODULE	005F	1 0	1 0	TRANS	2	
U10-9	PROBE	002A	1 0	1 0	TRANS	5	
U10-9	I/O MODULE	002A	1 0	1 0	TRANS	5	
U10-12	PROBE	008B	1 0	1 0	TRANS	5	
U10-12	I/O MODULE	008B	1 0	1 0	TRANS	5	
U10-15	PROBE	005F	1 0	1 0	TRANS	2	
U10-15	I/O MODULE	005F	1 0	1 0	TRANS	2	
U10-16	PROBE	008B	1 0	1 0	TRANS	5	
U10-16	I/O MODULE	008B	1 0	1 0	TRANS	5	
U10-19	PROBE	000A	1 0	1 0	TRANS	6	
U10-19	I/O MODULE	000A	1 0	1 0	TRANS	6	
U20-6	I/O MODULE	0000	1 0	0	TRANS	2	
U20-7	I/O MODULE	00FE	1	1	TRANS	0	
U20-9	I/O MODULE	0000	1 0	0	TRANS	2	
U20-15	PROBE	00FE	1 0	1	TRANS	2	
U20-15	I/O MODULE	00FE	1 0	1	TRANS	2	
R33-1	PROBE	00FE	1	1	TRANS	0	
R33-1	I/O MODULE	00FE	1	1	TRANS	0	
U5-11	I/O MODULE	00AB	1 0		TRANS		

Figure 4-123: Response File *(interrupt)*

Summary of Complete Solution for Interrupt Circuit 4.13.7.

The entire set of programs and files needed to test and GFI troubleshoot the Interrupt Circuit functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Interrupt Circuit)

Programs (PROGRAM):

TST_INTRPT	Functional Test	Section 4.13.5
CTRL_OUT3	Stimulus Program	Figure 4-103
INTERRUPT	Stimulus Program	Figure 4-122
DECODE	Stimulus Program	Figure 4-108

Stimulus Program Responses (RESPONSE):

CTRL_OUT3	Figure 4-104
INTERRUPT	Figure 4-123
DECODE	Figure 4-109

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

(This page is intentionally blank.)

READY CIRCUIT FUNCTIONAL BLOCK 4.14.

Introduction to Ready Circuits 4.14.1.

Some peripheral components have different (slower) timing than the microprocessor. To accommodate these components, wait states (extra clock cycles) are added to the read and write bus cycles. The number of wait states inserted is typically controlled by an input to the microprocessor called Wait, Ready, or DTACK; in this discussion, we will call it the Ready signal.

Many microprocessor systems have a circuit that generates the Ready signal in response to the selection of a peripheral component. The circuit (Figure 4-124) typically consists of a counter and/or a state machine that uses the microprocessor clock. The inputs to the state machine include a strobe signal from the microprocessor (to indicate that a bus cycle has started) and the various decoder outputs that select the components needing wait states.

In a given bus cycle, the state machine typically recognizes the assertion of the microprocessor strobe signal, and looks at the decoder signals to determine which component is being selected. The state machine then asserts Ready for the appropriate number of clock cycles.

Considerations for Testing and Troubleshooting 4.14.2.

Ready circuits often involve multiple feedback loops between the microprocessor and the ROM, RAM timing, and video control circuits. Since these feedback loops may need to remain unbroken while testing memory and/or video circuits, the Ready circuit is tested separately. Here is a good way test the Ready circuit:

1. Break the feedback loop by overdriving the lines that form the feedback loop.

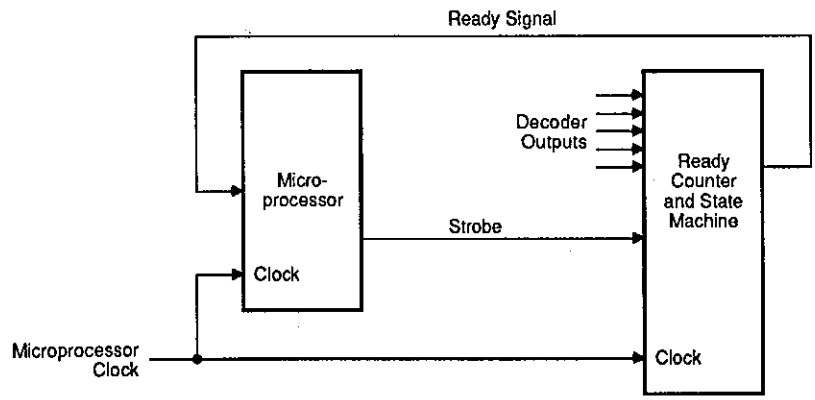


Figure 4-124: Typical Ready Circuit

2. Exercise the rest of the inputs using microprocessor reads and writes.

3. Measure the output of the loop.

A second approach is to use one I/O module to overdrive all the inputs and another I/O module (or another clip on the same I/O module) to measure the Ready output to the microprocessor.

Test each IC in the circuit individually, using the following procedure:

1. Clip the I/O module onto the IC.

2. Synchronize and arm the I/O module (see the *Technical User's Manual* for this procedure).

3. Run a stimulus procedure to make each output go high and low (this may mean overdriving another part of the circuit with another I/O module clip).

4. Use the SHOW I/O MOD command on the I/O MOD key (operator's keypad) to observe signatures on each pin of the IC.

5. Write down the signatures gathered from each pin on the IC, both inputs and outputs.

Compare the signatures gathered on the suspect UUT to those from a known-good UUT to determine which pins are bad.

Test the timing properties of the state machine that actually generates the Ready signal. You can do this with the external Start, Stop, and Clock lines on the I/O module or clock module to begin timing the wait states. Connect the external Clock line to the Ready-circuit's clock input (the microprocessor clock). Connect the Start line to the signal that starts the wait state generation. Set the Stop count to the proper number of clock cycles to verify that the wait state becomes active at the proper time. If the Stop count is set properly, decreasing its value by 1

from the proper value should show that the wait state does not become active and using the proper value should show that the wait state is active.

Again, compare the responses gathered on the suspect UUT to those from a known-good UUT to determine which pins are bad.

If the outputs of the ICs are bad and the inputs are good, suspect the IC and/or suspect shorts on the output signal paths. If the inputs are bad as well, trace back toward the microprocessor. If your UUT has address latches or buffers, perform a similar test on them.

You may need to disable the Ready input to the pod and turn reporting of forcing lines off when troubleshooting this circuit. If the Ready input to the pod is enabled, and Ready is not asserted for a long enough time due to testing operations, the pod may timeout if it is being used in the stimulus operation. Section 4.15.4, "Forcing Lines", in this manual describes how to disable the Ready input to the pod.

Ready Circuit Example 4.14.3.

The Ready Circuit for the Demo/Trainer UUT is shown in Figure 4-125. The microprocessor does not complete the current bus cycle until an active Ready signal (a low) is received from the Ready Circuit. Any circuit addressed to be read by the microprocessor must return such a Ready signal. Some circuits (ROM0, ROM1, and Interrupt) set SRDY low right away and the read is completed on the next clock cycle. Other circuits (Parallel I/O, Serial I/O, and Video Control) cannot match the speed of the microprocessor and add three wait states for proper timing. In addition, Dynamic RAM Timing may insert wait states in order to delay until RAM refresh finishes, and Video RAM may insert wait states to synchronize the microprocessor with video scan sequences.

The microprocessor drives address lines, which go to address decoding, and the outputs of address decode are inputs to the

Ready Circuit. The output of the Ready Circuit is an input to the microprocessor, which forms a feedback loop. The pod is able to break this feedback loop by ignoring and disabling the Ready input.

The Ready Circuit has a second, more troublesome feedback loop. The Ready output, U1-4, feeds back as an input to the Ready Circuit at U4-12. This second feedback loop must be broken in order to perform testing or troubleshooting on the Ready Circuit.

Keystroke Functional Test 4.14.4.

The functional test for the Ready Circuit uses two I/O module clips. One clip is used for measurement and the other clip is used to overdrive Ready Circuit inputs (to break the Ready Circuit feedback loop).

In the following procedure use one clip module to measure U1-4, U4-6, and U17-11 outputs. Use the second clip module as prompted by the program.

Part A:

1. Use a 20-pin clip module on side A of I/O module 1 and a 14-pin clip module on side B as the second clip of I/O module 1 to check the Ready Circuit output.

2. Use the EXEC and I/O MOD keys with the commands below for U1 and U4. The correct measurements for each pin are shown in the response table of Figure 4-125.

```
EXECUTE UUT DEMO PROGRAM READY_1
```

The program will prompt:

```
Enter ref name (Choose U1, U4, U14 OR U15)
```

Type in U1 and press the ENTER key.

Follow the instructions to clip U1 and press the Ready button on the clip module. Then clip U4 and press the Ready button on its clip module.

```
SHOW I/O MOD 1 PIN 4 CAPTURED RESPONSES
SHOW I/O MOD 1 PIN 26 CAPTURED RESPONSES
```

NOTE

The SHOW command requires a clip module pin number rather than a part pin number. This requires you to translate part pin numbers to clip module pin numbers (see Appendix B of the Technical User's Manual). For your convenience, this translation has been done for you in this example, and the results are shown in the "I/O MOD PIN" column of the response table in Figure 4-125.

Part B:

1. Use a 14-pin clip module on side B of I/O module 1 to check the Ready Circuit.

2. Use the EXEC and I/O MOD keys with the commands below for U4. The correct measurement for this step is shown in response table #1 of Figure 4-126.

   ```
   EXECUTE UUT DEMO PROGRAM READY_2
   ```

   The program will prompt:

   ```
   Enter ref name (Choose U1, U4, U5, U6 or U17)
   ```

   Type in U4 and press the ENTER key.

   Follow the instructions to clip U4 and press the Ready button on the clip module.

```
SHOW I/O MOD 1 PIN 26 CAPTURED RESPONSES
```

3. Use a 14-pin clip module on side A of I/O module 1 to check the Ready Circuit.

4. Use the EXEC and I/O MOD keys with the commands below for U4. The correct measurement for this step is shown in response table #2 of Figure 4-126.

```
EXECUTE UUT DEMO PROGRAM READY_3
```

The program will prompt:

```
Enter ref name  (Choose U1, U4, U5 or U6)
```

Type in U4 and press the ENTER key.

Follow the instructions to clip U4 and press the Ready button on the clip module.

```
SHOW I/O MOD 1 PIN 26 CAPTURED RESPONSES
```

Part C:

1. Use a 14-pin clip module on side A of I/O module 1 and a 20-pin clip module on side B as the second clip of I/O module 1 to check the Ready Circuit.

2. Use the EXEC and I/O MOD keys with the commands below for U5. The correct measurement for each pin is shown in the response table of Figure 4-127.

```
EXECUTE UUT DEMO PROGRAM READY_4
```

The program will prompt:

```
Enter ref name  (Choose U4, U5 or U17)
```

Type in U5 and press the ENTER key.

Follow the instructions to clip U5 and press the Ready button on the clip module.

Then clip U17 using the second clip module and press its Ready button.

```
SHOW I/O MOD 1 PIN 3 CAPTURED RESPONSES
```

Part D:

1. Use a 20-pin clip module on side A of I/O module 1 to check the Ready Circuit I/O wait state generator.

2. Use the EXEC and I/O MOD keys with the commands below for U17. The correct measurement for this step is shown in response table #1 of Figure 4-128.

```
EXECUTE UUT DEMO PROGRAM READY_5
```

The program will prompt:

```
Enter ref name  (Choose U5 or U17)
```

Type in U17 and press the ENTER key.

Follow the instructions to clip U17 and press the Ready button on the clip module.

```
SHOW I/O MOD 1 PIN 17 CAPTURED RESPONSES
```

3. Use a 20-pin clip module on side A of I/O module 1 to check the Ready Circuit I/O wait state generator.

4. Use the EXEC and I/O MOD keys with the commands below for U17. The correct measurement responses for each step are shown in response table #2 of Figure 4-128.

```
EXECUTE UUT DEMO PROGRAM READY_6
```

The program will prompt:

```
Enter ref name  (Choose U5 or U17)
```

Type in U17 and press the ENTER key.

Follow the instructions to clip U17 and press the Ready button on the clip module.

```
SHOW I/O MOD 1 PIN 17 CAPTURED RESPONSES
```

Keystroke Functional Test (Part A)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] U4-4 U4-5	I/O MOD U1-4 U4-6

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
READY	U1-4	4	0015
SRDY	U4-6	26	0015

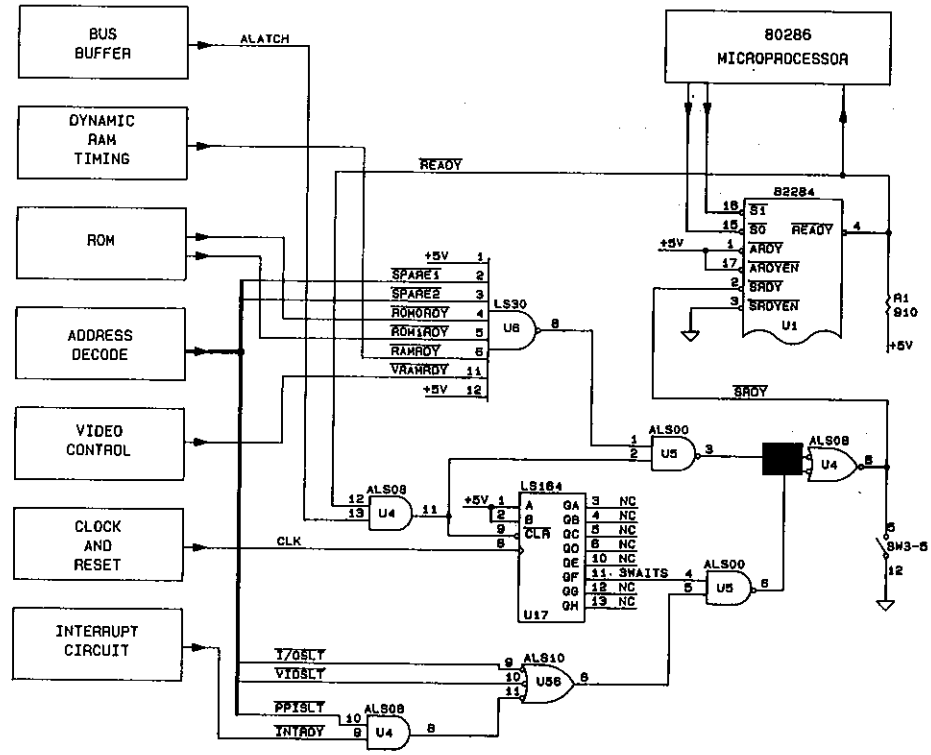


Figure 4-125: Ready Circuit Functional Test (Part A)

Keystroke Functional Test (Part B)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] TEST ACCESS SOCKET [redacted] U4-11	I/O MOD U4-6

RESPONSE TABLE #1

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE	ASYNC LEVEL
SRDY	U4-6	26	0 0 0 0	IO

RESPONSE TABLE #2

SIGNAL	PART PIN	I/O MOD PIN	ASYNC LEVEL	TRANS COUNT
SRDY	U4-6	26	0 1	3

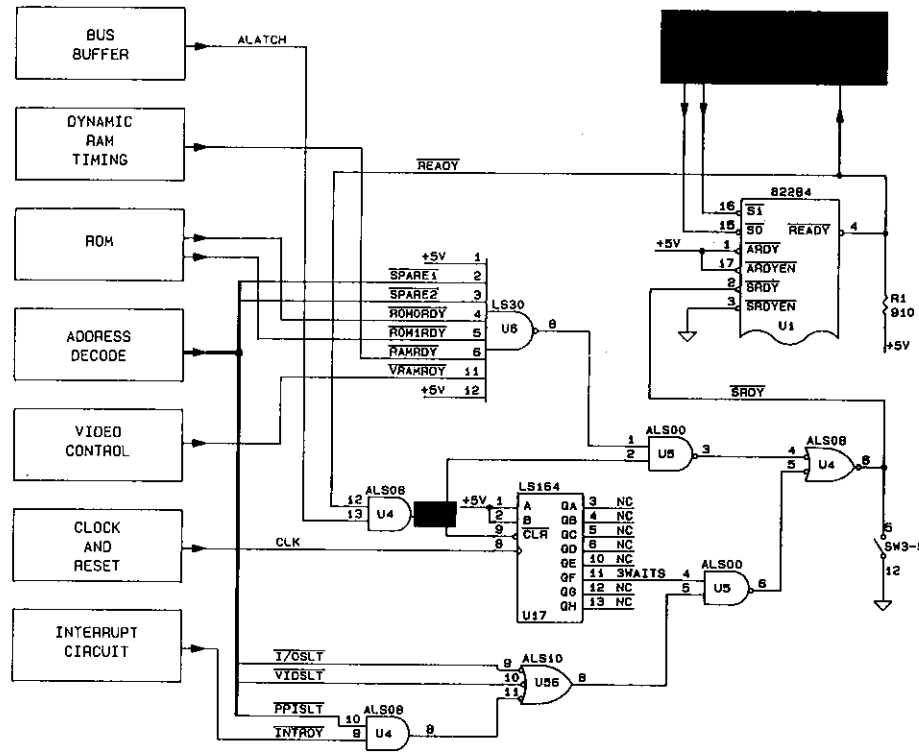


Figure 4-126: Ready Circuit Functional Test (Part B)

Keystroke Functional Test (Part C)

CONNECTION TABLE

[redacted]	MEASUREMENT
[redacted] U5-1 U17-9	I/O MOD U5-3

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
—	U5-3	3	0 0 0 A

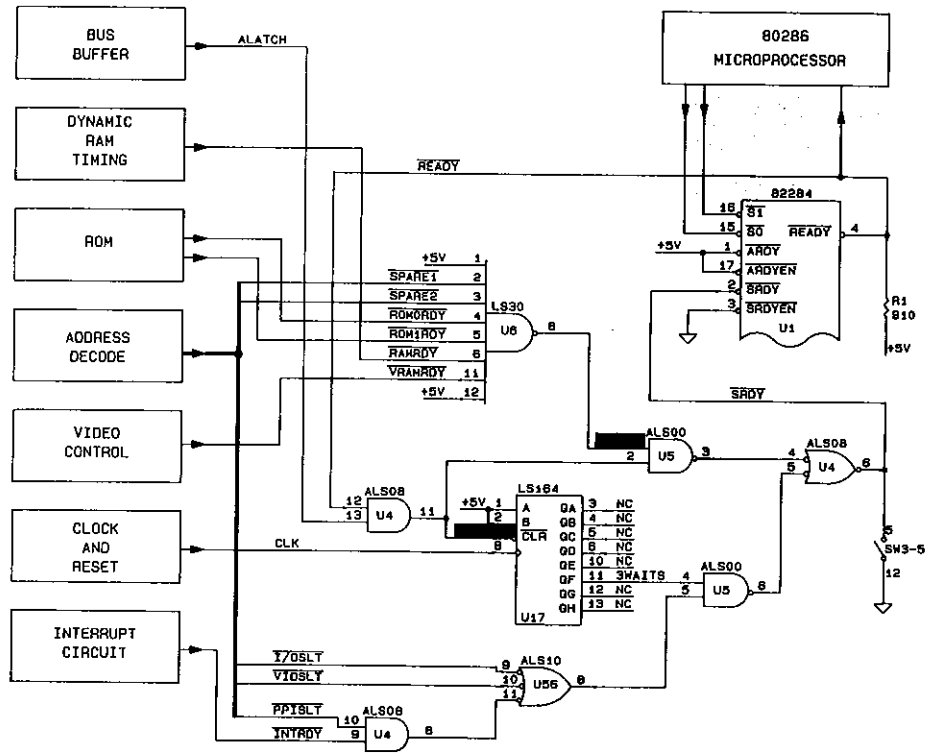


Figure 4-127: Ready Circuit Functional Test (Part C)

Keystroke Functional Test (Part D)

CONNECTION TABLE

[redacted]	MEASUREMENT CONTROL	MEASUREMENT
[redacted]	I/O MOD	I/O MOD
U17-9	CLOCK U1-10 START U4-11	U17-11

RESPONSE TABLE #1

SIGNAL	PART PIN	I/O MOD PIN	ASYNC LEVEL	TRANS COUNT
3WAITS	U17-11	17	01	1

RESPONSE TABLE #2

SIGNAL	PART PIN	I/O MOD PIN	TRANS COUNT
3WAITS	U17-11	17	0

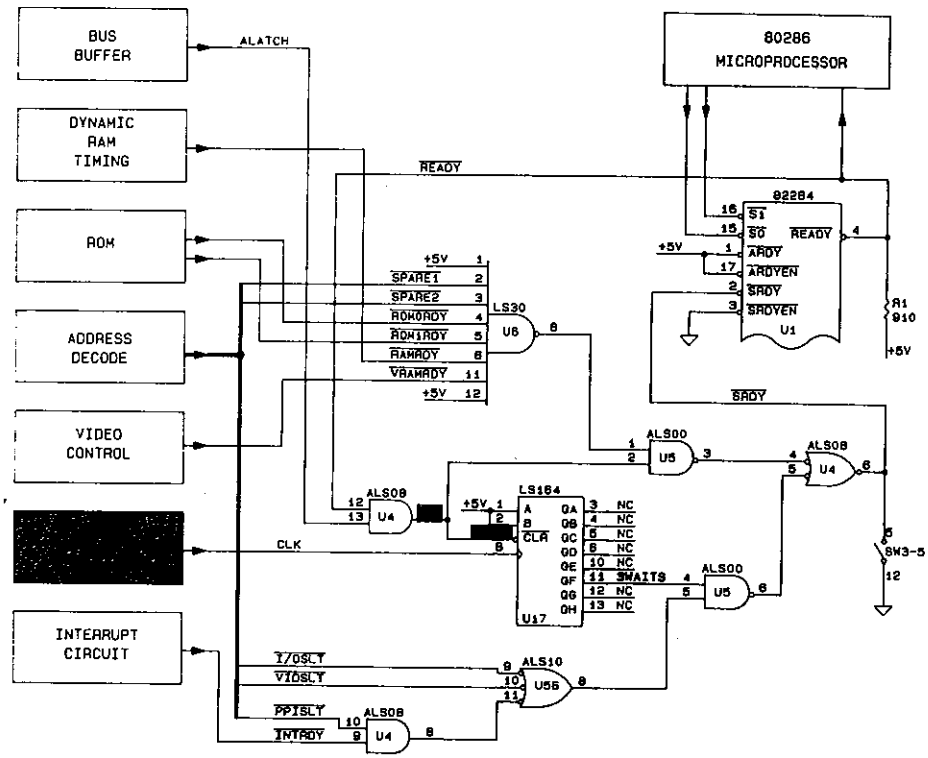


Figure 4-128: Ready Circuit Functional Test (Part D)

Programmed Functional Test 4.14.6.

The *tst_ready* program is the programmed functional test for the Ready Circuit functional block. This program checks the Ready circuit using the *gfi test* command. If the *gfi test* command fails, the *abort_test* program is executed and GFI troubleshooting begins. (See the Bus Buffer functional block for a discussion of the *abort_test* program).

The *gfi test* command executes a number of stimulus programs. The *ready_1*, *ready_2*, *ready_3*, and *ready_4* stimulus programs overdrive nodes in order to break the feedback loop in the Ready circuit. These programs will ask the operator to use a second clip on a second component so that the circuit can be overdriven.

```
program tst_ready

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the READY functional block.                         !
!                                                                        !
! This program tests the READY functional block of the Demo/Trainer.     !
! The gfi test command and I/O module are used to perform the test. The  !
! ready test involves overdriving components to break the feedback loop  !
! in the ready partition.  Two I/O module clips are required; one for    !
! measurement and one for stimulus (overdriving).                        !
!                                                                        !
! TEST PROGRAMS CALLED:                                                  !
!    abort_test (ref-pin)                  If gfi has an accusation      !
!                                          display the accusation else   !
!                                          create a gfi hint for the     !
!                                          ref-pin and terminate the test!
!                                          program (GFI begins trouble-  !
!                                          shooting).                    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   if (gfi status "U1-4") = "untested" then
      print "\nl\nlTESTING READY CIRCUIT"

      podsetup 'enable ~ready' "off"
      podsetup 'report forcing' "off"

      if (gfi status "U1-4") = "untested" then gfi test "U1-4"
      if (gfi status "U1-4") = "bad" or (gfi status "U1-2") = "bad" or
                                        (gfi status "U1-3") = "bad" then
         abort_test("U1-4")
      else
         print "READY CIRCUIT PASSES"
      end if
   end if
end program
```

Stimulus Programs and Responses 4.14.7.

Figure 4-129 is the stimulus program planning diagram for the Ready Circuit functional block. The *ready_1, ready_2, ready_3,* and *ready_4* stimulus programs use one clip for measurement and a second clip to overdrive the Ready circuit in order to break the feedback loop in this circuit. *ready_5* and *ready_6* provide stimulus to measure the operation of the I/O ready generator, U17. These two stimulus programs count how many 8 Mhz clocks occur during the wait state generated by U17.

The steps to break the Ready feedback loop to diagnose a fault are shown below:

1. Overdrive inputs U4-4 and U4-5. Then measure outputs U4-6 and U1-4. The 82284 chip (U1) synchronizes the Ready output (U4-6) to the microprocessor read/write cycles. This requires the *ready_1* stimulus program to output the level, allow enough time for the signal to get synchronized, then check the level at the output U1-4.

2. Finish breaking the Ready signal feedback loop by overdriving inputs U4-12 and U4-13, then measure the outputs U4-11, U5-3, and U4-6. In order to measure U5-3 and U4-6, the other inputs U5-1 and U4-5 must be held high so the signals will flow through the AND gates. The *ready_4* stimulus program performs this step.

3. Hold the node with output source U4-11 high. This allows signals from U6 to flow through U5-3 to U4-6. At the same time, holding U4-11 high causes output U17-11 to stabilize at a high state, allowing signals from U56 to ripple through U5-6 to U4-6. Now use the pod to exercise the Ready Circuit inputs that are driven by the Address Decode functional block. The *ready_2* stimulus program performs this sequence for all components that can be forced to use zero wait states. It does this by disabling U17 (all

components except RAM and Video RAM). Since the pod has turned ~READY ENABLE OFF, the pod generates a sync pulse with zero wait states. Because the RAM and Video RAM return wait states, taking signature measurements on RAM and Video RAM will turn out to be unstable. To solve this problem, *ready_2* accesses all components except RAM and Video RAM. Then the *ready_3* stimulus program performs a similar operation, but exercises only RAM and Video RAM. *ready_3* responses are characterized by asynchronous level history and transition counts to allow the RAM and Video RAM wait state signals to be measured.

4. Measure the I/O component wait state generator, U17. The Clear input at U17-9 is toggled low. At the same time a measurement using external Clock (and Start) is made. The External Clock line is connected to the 8 MHz clock CLK and the Start line is connected to the node which includes U17-9. A Stop Count is set and transition counts and level history are measured. The *ready_6* stimulus program uses a Stop Count of four clocks and the response is expected to be low level history and zero transitions, indicating that the wait state output was low for at least four clocks. The *ready_5* stimulus program uses a Stop Count of six clocks. In this case, a response of high and low level history is expected, and a transition count of 1 is expected. These results indicate that the wait state finished within six clock cycles.

Advice for Making GFI Work in the Presence of Ready Faults

When a Ready fault exists, a forcing-line fault condition will be generated. However, the pod must ignore the Ready forcing-line fault condition so that the stimulus program will execute completely. Otherwise, a fault condition would be generated and GFI would terminate. To turn this report off, a SETUP REPORT FORCING ~READY OFF command can be

performed. When this is done, the pod will continue to respond to the Ready signal, but will not generate a fault message. If the Ready signal is stuck high, the pod will cause the 9100A/9105A to generate a pod timeout fault condition. To cure this, a SETUP ENABLE ~READY OFF command is performed. At this point, GFI will work properly and Ready problems can be isolated to the failing component or node.

More generally, GFI works best if every stimulus program turns all reporting conditions off. In addition, those stimulus programs that create activity in the kernel area, may need to turn off Enable Ready. All Demo/Trainer UUT stimulus programs related to the address bus, data bus, control signals, address decoding, interrupts, and ready circuitry turn the Ready Enable off at the beginning of the stimulus program and the turn Ready Enable back on at the end of the program.

One more note: the 80286 microprocessor uses a separate bus controller that has no feedback lines to the microprocessor. When the pod disables the Ready input and performs zero wait state operations regardless of the Ready input, the bus controller can get out of synchronization from the pod and may get confused. When this happens, an *enabled_line_timeout* fault condition is generated. The solution is to provide a handler for that fault condition in each stimulus program that enables and disables Ready. The handler for the fault condition should call a program which performs a recovery procedure. The recovery procedure depends on the UUT. Usually, forcing the Ready line active or performing a Reset will recover synchronization. Or, by disabling Ready and then performing a read or write in memory space followed by enabling Ready may recover synchronization of the 80286 pod and the bus controller. Most other microprocessors do not have this problem.

Stimulus Program Planning

PROGRAM: READY_1
OVERDRIVES U4-6 TO CHECK THE SYNCHRONIZED READY OUTPUT
MEASUREMENT AT: U1-4 U4-6

PROGRAM: READY_2
OVERDRIVES THE NODE AT U4-11 AND ALSO EXERCISES THE READY RETURN LINES (EXCEPT VRAM AND VRAMRDY)
MEASUREMENT AT: U4-6,8 U5-3,6 U6-8 U56-8

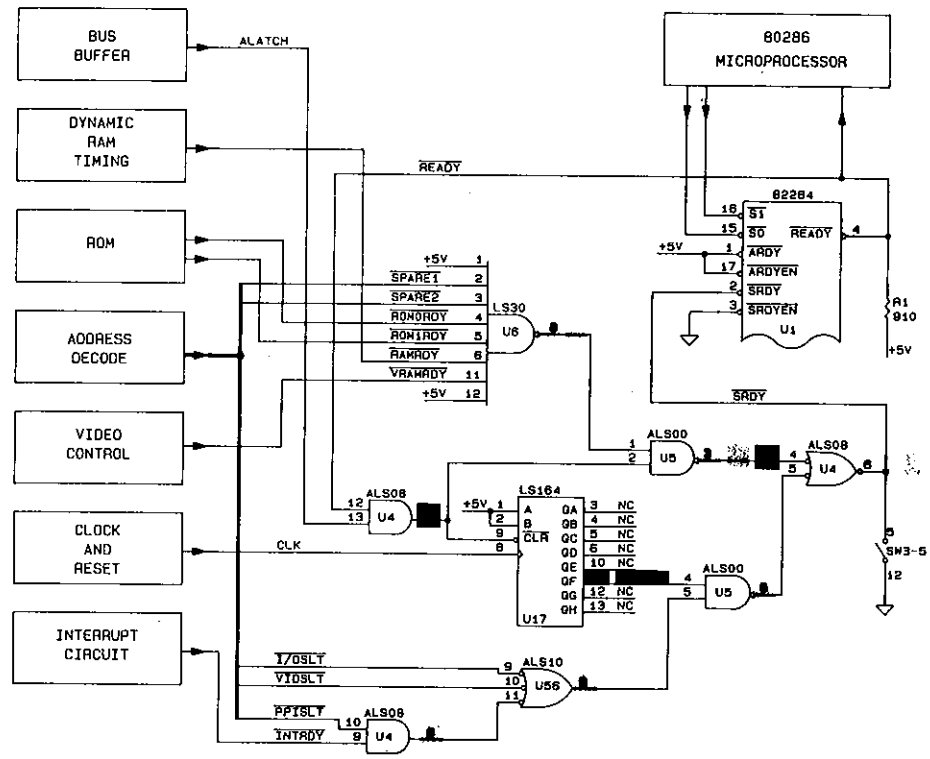


Figure 4-129: Ready Circuit Stimulus Program Planning

```
program ready_1

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM overdrives U4 in ready circuit.                            !
!                  Characterizes U4-6 and U1-4.                               !
!                                                                             !
! Stimulus programs and response files are used by GFI to backtrace          !
! from a failing node.  The stimulus program must create repeatable UUT      !
! activity and the response file contains the known-good responses for       !
! the outputs in the UUT that are stimulated by the stimulus program.        !
!                                                                             !
! This stimulus program is one of the programs which creates activity        !
! in the kernel area of the UUT.  These programs create activity with        !
! or without the ready circuit working properly.  Because of this, all       !
! the stimulus programs in the kernel area must disable the READY input      !
! to the pod, then perform the stimulus, then re-enable the READY input      !
! to the pod.  The 80286 microprocessor has a separate bus controller;       !
! for this reason, disabling ready and performing stimulus can get the       !
! bus controller out of synchronization with the pod.  Two fault             !
! handlers trap pod timeout conditions that indicate the bus controller      !
! is out of synchronization.  The recover() program is executed to           !
! resynchronize the bus controller and the pod.                              !
!                                                                             !
! TEST PROGRAMS CALLED:                                                       !
!    recover     ()                        The 80286 microprocessor has a!
!                                          bus controller that is totally!
!                                          separate from the pod.  In    !
!                                          some cases the pod can get out!
!                                          of sync with the bus control- !
!                                          ler.  The recover program     !
!                                          resynchronizes the pod and the!
!                                          bus controller.               !
!                                                                             !
! GRAPHICS PROGRAMS CALLED:                                                   !
!    (none)                                                                   !
!                                                                             !
! Global Variables Modified:                                                  !
!    recover_times                         Reset to Zero                      !
!                                                                             !
!                                                                             !
! Local Variables Modified:                                                   !
!    measure_dev                           Measurement device                 !
!    stimulus_dev                          Stimulus device (overdrives)       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-130: Stimulus Program *(ready_1)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle
handle pod_timout_no_clk
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine measurement device

   if (gfi control) = "yes" then
      measure_dev = gfi device
      measure_ref = gfi ref
   else
      print "Enter reference name of part to measure:"
      print "    (Chose U1, U4, U14 or U15)"
      measure_ref = "" \ input measure_ref
      if measure_ref <> "U14" then
         measure_dev = clip ref measure_ref
      else
         probe ref "U14-63" \ measure_dev = "/probe"
      end if
   end if

! Determine stimulus device

   if measure_ref = "U4" then
      stimulus_dev = measure_dev
   else
      print "\07\1B[2J\1B(201\1B[3;1f        USING \1B[7mSECOND\1B[0m CLIP."
      stimulus_dev = clip ref "U4"
      print "\1B[20h"
   end if
   print "Stimulus Program READY_1"
```

(continued on the next page)

Figure 4-130: Stimulus Program *(ready_1)* - *continued*

```
! Setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'standby function off'
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   reset device measure_dev
   reset device stimulus_dev
   sync device measure_dev, mode "int"

! Perform Stimulus

   arm device measure_dev
      writepin device "U4", pin 4, level "1", mode "latch"
      writepin device "U4", pin 5, level "1", mode "latch"
      strobeclock device measure_dev
      writepin device "U4", pin 4, level "0", mode "latch"
      writepin device "U4", pin 5, level "1", mode "latch"
      strobeclock device measure_dev
      writepin device "U4", pin 4, level "1", mode "latch"
      writepin device "U4", pin 5, level "1", mode "latch"
      strobeclock device measure_dev
      writepin device "U4", pin 4, level "1", mode "latch"
      writepin device "U4", pin 5, level "0", mode "latch"
      strobeclock device measure_dev
      writepin device "U4", pin 4, level "1", mode "latch"
      writepin device "U4", pin 5, level "1", mode "latch"
      strobeclock device measure_dev
   readout device measure_dev

   clearoutputs device stimulus_dev
   podsetup 'standby function on'
   podsetup 'enable ~ready' "on"

end program
```

Figure 4-130: Stimulus Program *(ready_1) - continued*

STIMULUS PROGRAM NAME: READY_1
DESCRIPTION: SIZE: 94 BYTES

Node Signal Src	Learned With	SIG	Response Data Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U4-6	I/O MODULE	0015	1	0	TRANS		
U1-4	PROBE	0015	1	0	TRANS		
U1-4	I/O MODULE	0015	1	0	TRANS		

Figure 4-131: Response File *(ready_1)*

```
program ready_2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM overdrives U4 in ready circuit.                         !
!                  Characterizes U4-6 and U1-4.                            !
!                                                                          !
! Stimulus programs and response files are used by GFI to backtrace       !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.    !
!                                                                          !
! This stimulus program is one of the programs which creates activity    !
! in the kernel area of the UUT.  These programs create activity with    !
! or without the ready circuit working properly.  Because of this, all  !
! the stimulus programs in the kernel area must disable the READY input !
! to the pod, then perform the stimulus, then re-enable the READY input !
! to the pod.  The 80286 microprocessor has a separate bus controller;  !
! for this reason, disabling ready and performing stimulus can get the  !
! bus controller out of synchronization with the pod.  Two fault         !
! handlers trap pod timeout conditions that indicate the bus controller !
! is out of synchronization.  The recover() program is executed to       !
! resynchronize the bus controller and the pod.                           !
!                                                                          !
! TEST PROGRAMS CALLED:                                                    !
!    recover     ()                      The 80286 microprocessor has a!
!                                        bus controller that is totally!
!                                        separate from the pod.  In     !
!                                        some cases the pod can get out!
!                                        of sync with the bus control- !
!                                        ler.  The recover program      !
!                                        resynchronizes the pod and the!
!                                        bus controller.                !
!                                                                          !
! GRAPHICS PROGRAMS CALLED:                                                !
!    (none)                                                                !
!                                                                          !
! Global Variables Modified:                                               !
!    recover_times                       Reset to Zero                     !
!                                                                          !
!                                                                          !
! Local Variables Modified:                                                !
!    measure_dev                         Measurement device                !
!    stimulus_dev                        Stimulus device (overdrives)     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-132: Stimulus Program *(ready_2)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle
handle pod_timout_no_clk
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine measurement device

   if (gfi control) = "yes" then
      measure_dev = gfi device
      measure_ref = gfi ref
   else
      print "Enter reference name of part to measure:"
      print "    (Chose U1, U4, U5, U6, U56 or U17)"
      measure_ref = "" \ input measure_ref
      measure_dev = clip ref measure_ref
   end if

! Determine stimulus device

   if measure_ref = "U1" then
      print "\07\1B[2J\1B[201\1B[3;1f          USING \1B[7mSECOND\1B[0m CLIP."
      stimulus_dev = clip ref "U4"
      print "\1B[20h"
   else
      stimulus_dev = measure_dev
   end if
   print "Stimulus Program READY_2"

! Setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   io_byte = getspace space "i/o", size "byte"
   mem_word = getspace space "memory", size "word"
```

(continued on the next page)

Figure 4-132: Stimulus Program *(ready_2)* - continued

```
   reset device measure_dev
   reset device stimulus_dev
   sync device measure_dev, mode "pod"
   sync device "/pod", mode "data"
   old_cal = getoffset device measure_dev
   setoffset device measure_dev, offset (1000000 - 56)

   if measure_ref = "U5" then
      writepin device "U5", pin 2,  level "1", mode "latch"
      writepin device "U5", pin 4,  level "1", mode "latch"
   else if measure_ref = "U4" or measure_ref = "U1" then
      writepin device "U4", pin 11, level "1", mode "latch"
   end if

! Stimulate ICs and capture response.

   arm device measure_dev          ! Start response capture.
      setspace (mem_word)
      read addr $30000             ! IPOLL
      read addr $40000             ! SPARE1
      read addr $50000             ! SPARE2
      read addr $E0000             ! ROM0
      read addr $F0000             ! ROM1
      setspace (io_byte)
      read addr 0                  ! VIDSLT
      read addr $2000              ! I/OSLT
      read addr $4000              ! PPISLT
   readout device measure_dev      ! End response capture.

   if stimulus_dev <> "/probe" then clearoutputs device stimulus_dev
   setoffset device measure_dev, offset old_cal
   podsetup 'enable ~ready' "on"

end program
```

Figure 4-132: Stimulus Program *(ready_2) - continued*

STIMULUS PROGRAM NAME: READY_2
DESCRIPTION: SIZE: 143 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Response Data: Clk LVL	Response Data: Counter Mode	Counter Range	Priority Pin
U4-6	I/O MODULE	0000	1	0	TRANS		
U4-8	PROBE	007E	1	0	TRANS		
U4-8	I/O MODULE	007E	1	0	TRANS		
U5-3	I/O MODULE	0086	1	0	TRANS		
U5-6	I/O MODULE	0078	1	0	TRANS		
U56-8	PROBE	0086	1	0	TRANS		
U56-8	I/O MODULE	0086	1	0	TRANS		
U6-8	I/O MODULE	0078	1	0	TRANS		

Figure 4-133: Response File *(ready_2)*

```
program ready_3

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM toggles ready circuit inputs which generate           !
!                  wait states.                                          !
!                                                                        !
! Stimulus programs and response files are used by GFI to backtrace     !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                        !
! This stimulus program is one of the programs which creates activity   !
! in the kernel area of the UUT.  These programs create activity with   !
! or without the ready circuit working properly.  Because of this, all  !
! the stimulus programs in the kernel area must disable the READY input !
! to the pod, then perform the stimulus, then re-enable the READY input !
! to the pod.  The 80286 microprocessor has a separate bus controller;  !
! for this reason, disabling ready and performing stimulus can get the  !
! bus controller out of synchronization with the pod.  Two fault        !
! handlers trap pod timeout conditions that indicate the bus controller !
! is out of synchronization.  The recover() program is executed to      !
! resynchronize the bus controller and the pod.                         !
!                                                                        !
! TEST PROGRAMS CALLED:                                                  !
!    recover     ()                     The 80286 microprocessor has a!
!                                       bus controller that is totally!
!                                       separate from the pod.  In    !
!                                       some cases the pod can get out!
!                                       of sync with the bus control- !
!                                       ler.  The recover program     !
!                                       resynchronizes the pod and the!
!                                       bus controller.               !
!                                                                        !
! GRAPHICS PROGRAMS CALLED:                                              !
!    (none)                                                              !
!                                                                        !
! Global Variables Modified:                                             !
!    recover_times                      Reset to Zero                   !
!                                                                        !
! Local Variables Modified:                                              !
!    measure_dev                        Measurement device              !
!    stimulus_dev                       Stimulus device (overdrives)    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-134: Stimulus Program *(ready_3)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle
handle pod_timout_no_clk
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine measurement device

   if (gfi control) = "yes" then
      measure_dev = gfi device
      measure_ref = gfi ref
   else
      print "Enter reference name of part to measure:"
      print "    (Chose U1, U4, U5 or U6)"
      measure_ref = "" \ input measure_ref
      measure_dev = clip ref measure_ref
   end if

! Determine stimulus device

   if measure_ref = "U1" then
      print "\07\1B[2J\1B[201\1B[3;1f        USING \1B[7mSECOND\1B[0m CLIP."
      stimulus_dev = clip ref "U4"
      print "\1B[20h"
   else
      stimulus_dev = measure_dev
   end if
   print "Stimulus Program READY_3"
```

(continued on the next page)

Figure 4-134: Stimulus Program *(ready_3) - continued*

```
! Setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'standby function off'
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   io_byte = getspace space "i/o", size "byte"
   mem_word = getspace space "memory", size "word"
   reset device measure_dev
   reset device stimulus_dev
   sync device measure_dev, mode "pod"
   sync device "/pod", mode "data"
   old_cal = getoffset device measure_dev
   setoffset device measure_dev, offset (1000000 - 56)

   if measure_ref = "U5" then
      writepin device "U5", pin 2,  level "1", mode "latch"
      writepin device "U5", pin 4,  level "1", mode "latch"
   else if measure_ref = "U4" or measure_ref = "U1" then
      writepin device "U4", pin 11, level "1", mode "latch"
   end if

! Stimulate ICs and capture response.

   arm device measure_dev           ! Start response capture.
      setspace (mem_word)
      read addr 0                   ! RAM0
      read addr $10000              ! RAM1
      write addr $20000, data 0     ! VRAM (write only)
   readout device measure_dev       ! End response capture.

   clearoutputs device stimulus_dev
   setoffset device measure_dev, offset old_cal
   podsetup 'standby function on'
   podsetup 'enable ~ready' "on"

end program
```

Figure 4-134: Stimulus Program *(ready_3) - continued*

STIMULUS PROGRAM NAME: READY_3
DESCRIPTION:

SIZE: 112 BYTES

Node Signal Src	Learned With	SIG	Response Data: Async LVL	Clk LVL	Counter Mode	Counter Range	Priority Pin
U4-6	I/O MODULE		1	0	TRANS	3	
U5-3	I/O MODULE		1	0	TRANS	3	
U6-8	I/O MODULE		1	0	TRANS	3	

Figure 4-135: Response File *(ready_3)*

```
program ready_4

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM overdrives U4 in ready circuit.                            !
!                  Characterizes U4-6 and U1-4.                               !
!                                                                             !
! Stimulus programs and response files are used by GFI to backtrace          !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                             !
! This stimulus program is one of the programs which creates activity  !
! in the kernel area of the UUT.  These programs create activity with  !
! or without the ready circuit working properly.  Because of this, all  !
! the stimulus programs in the kernel area must disable the READY input !
! to the pod, then perform the stimulus, then re-enable the READY input !
! to the pod.  The 80286 microprocessor has a separate bus controller;  !
! for this reason, disabling ready and performing stimulus can get the  !
! bus controller out of synchronization with the pod.  Two fault        !
! handlers trap pod timeout conditions that indicate the bus controller !
! is out of synchronization.  The recover() program is executed to      !
! resynchronize the bus controller and the pod.                         !
!                                                                             !
! TEST PROGRAMS CALLED:                                                       !
!    recover     ()                           The 80286 microprocessor has a!
!                                             bus controller that is totally!
!                                             separate from the pod.  In    !
!                                             some cases the pod can get out!
!                                             of sync with the bus control- !
!                                             ler.  The recover program     !
!                                             resynchronizes the pod and the!
!                                             bus controller.               !
!                                                                             !
! GRAPHICS PROGRAMS CALLED:                                                   !
!    (none)                                                                   !
!                                                                             !
! Global Variables Modified:                                                  !
!    recover_times                            Reset to Zero                 !
!                                                                             !
! Local Variables Modified:                                                   !
!    measure_dev                              Measurement device            !
!    stimulus_dev                             Stimulus device (overdrives)  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
```

(continued on the next page)

Figure 4-136: Stimulus Program *(ready_4)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle
handle pod_timout_no_clk
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine measurement device

   if (gfi control) = "yes" then
      measure_dev = gfi device
      measure_ref = gfi ref
   else
      print "Enter reference name of part to measure:"
      print "   (Chose U4, U5 or U17)"
      measure_ref = "" \ input measure_ref
      measure_dev = clip ref measure_ref
   end if

! Determine stimulus device

   if measure_ref = "U4" then
      print "\07\1B[2J\1B[201\1B[3;1f          USING \1B[7mSECOND\1B[0m CLIP."
      stimulus_dev = clip ref "U45"
   else if measure_ref = "U5" then
      print "\07\1B[2J\1B[201\1B[3;1f          USING \1B[7mSECOND\1B[0m CLIP."
      stimulus_dev = clip ref "U17"
   else if measure_ref = "U17" then
      print "\07\1B[2J\1B[201\1B[3;1f          USING \1B[7mSECOND\1B[0m CLIP."
      stimulus_dev = clip ref "U4"
   end if
   print "\1B[20h"
   print "Stimulus Program READY_4"
```

(continued on the next page)

Figure 4-136: Stimulus Program *(ready_4) - continued*

```
! Setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   reset device measure_dev
   reset device stimulus_dev
   sync device measure_dev, mode "int"
   sync device stimulus_dev, mode "int"

   if measure_ref = "U4" then
      storepatt device "U4", pin 12, patt "10111"
      storepatt device "U4", pin 13, patt "11101"
      storepatt device "U45", pin 6, patt "00000"
      storepatt device "U45", pin 3, patt "00000"
   else if measure_ref = "U5" then
      storepatt device "U5", pin 1,   patt "11111"
      storepatt device "U17", pin 9,  patt "10101"
   else if measure_ref = "U17" then
      storepatt device "U4", pin 12, patt "10111"
      storepatt device "U4", pin 13, patt "11101"
   end if

! Provide stimulus to UUT using I/O module to overdrive.

   arm device measure_dev
      if measure_ref = "U4" then
         writepatt device "U45,U4", mode "pulse"
      else if measure_ref = "U5" then
         writepatt device "U17,U5", mode "pulse"
      else if measure_ref = "U17" then
         writepatt device "U4", mode "pulse"
      end if
   readout device measure_dev

   podsetup 'enable ~ready' "on"
end program
```

Figure 4-136: Stimulus Program *(ready_4) - continued*

```
STIMULUS PROGRAM NAME: READY_4
DESCRIPTION:                                          SIZE:          78 BYTES

                    ------------------- Response Data --------------------
   Node              Learned                 Async Clk Counter                    Priority
Signal Src           With          SIG        LVL  LVL  Mode     Counter Range     Pin

U4-11                I/O MODULE    0015       1    0         TRANS
U5-3                 I/O MODULE    000A       1    0         TRANS
```

Figure 4-137: Response File *(ready_4)*

```
program ready_5

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM characterizes the ready circuit.                          !
!                                                                           !
! Stimulus programs and response files are used by GFI to backtrace         !
! from a failing node.  The stimulus program must create repeatable UUT     !
! activity and the response file contains the known-good responses for      !
! the outputs in the UUT that are stimulated by the stimulus program.       !
!                                                                           !
! This stimulus program is one of the programs which creates activity       !
! in the kernel area of the UUT.  These programs create activity with       !
! or without the ready circuit working properly.  Because of this, all      !
! the stimulus programs in the kernel area must disable the READY input     !
! to the pod, then perform the stimulus, then re-enable the READY input     !
! to the pod.  The 80286 microprocessor has a separate bus controller;      !
! for this reason, disabling ready and performing stimulus can get the      !
! bus controller out of synchronization with the pod.  Two fault            !
! handlers trap pod timeout conditions that indicate the bus controller     !
! is out of synchronization.  The recover() program is executed to          !
! resynchronize the bus controller and the pod.                             !
!                                                                           !
! TEST PROGRAMS CALLED:                                                     !
!    recover     ()                         The 80286 microprocessor has a!
!                                           bus controller that is totally!
!                                           separate from the pod.  In    !
!                                           some cases the pod can get out!
!                                           of sync with the bus control- !
!                                           ler.  The recover program     !
!                                           resynchronizes the pod and the!
!                                           bus controller.               !
!                                                                           !
!    check_meas (device, start, stop, clock, enable)                        !
!                                           Checks to see if the measure- !
!                                           ment is complete using the    !
!                                           TL/1 checkstatus command.  If !
!                                           the measurement times out then!
!                                           redisplay connect locations.  !
!                                                                           !
! GRAPHICS PROGRAMS CALLED:                                                 !
!    (none)                                                                 !
!                                                                           !
! Local Variables Modified:                                                 !
!    done                                   returned from check_meas()      !
!                                                                           !
! Global Variables Modified:                                                !
!    recover_times                          Reset to Zero                   !
!                                                                           !
! Local Variables Modified:                                                 !
!    measure_dev                            Measurement device              !
!    stimulus_dev                           Stimulus device (overdrives)    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

(continued on the next page)

Figure 4-138: Stimulus Program *(ready_5)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
declare numeric done = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      measure_dev = gfi device
      measure_ref = gfi ref
   else
      print "Enter reference name of part to measure:"
      print "    (Chose U5 or U17)"
      measure_ref = "" \ input measure_ref
      measure_dev = clip ref measure_ref
   end if
   print "Stimulus Program READY_5"

! Set addressing mode and setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'standby function off'
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   setspace( getspace( "i/o", "byte" ))
   reset device measure_dev
   sync device measure_dev, mode "ext"
   enable device measure_dev, mode "high"
   edge device measure_dev, start "+", stop "count", clock "-"
   stopcount device measure_dev, count 7
```

(continued on the next page)

Figure 4-138: Stimulus Program *(ready_5) - continued*

```
! Prompt user to connect external lines.

   if measure_ref = "U17" then
      connect device measure_dev, start "U4-11", clock "U1-10", common "gnd"
   else
      connect device measure_dev, start "U17-9", clock "U1-10", common "gnd"
   end if

! External lines determine measurement.

   loop until done = 1
      arm device measure_dev
         read addr 0
         done = check_meas(measure_dev,"U4-11", "*", "U1-10", "*")
      readout device measure_dev
   end loop

   clearoutputs device measure_dev
   podsetup 'standby function on'
   podsetup 'enable ~ready' "on"
end program
```

Figure 4-138: Stimulus Program *(ready_5) - continued*

```
STIMULUS PROGRAM NAME: READY_5
DESCRIPTION:                                          SIZE:          69 BYTES

                    ------------------- Response Data --------------------
   Node      Learned                    Async Clk Counter                    Priority
Signal Src    With        SIG            LVL  LVL  Mode     Counter Range     Pin

U17-11       I/O MODULE                   1  0      TRANS    1
```

Figure 4-139: Response File *(ready_5)*

```
program ready_6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! STIMULUS PROGRAM to wiggle all address lines from the uP.             !
!                                                                       !
! Stimulus programs and response files are used by GFI to backtrace     !
! from a failing node.  The stimulus program must create repeatable UUT !
! activity and the response file contains the known-good responses for  !
! the outputs in the UUT that are stimulated by the stimulus program.   !
!                                                                       !
! This stimulus program is one of the programs which creates activity   !
! in the kernel area of the UUT.  These programs create activity with   !
! or without the ready circuit working properly.  Because of this, all  !
! the stimulus programs in the kernel area must disable the READY input !
! to the pod, then perform the stimulus, then re-enable the READY input !
! to the pod.  The 80286 microprocessor has a separate bus controller;  !
! for this reason, disabling ready and performing stimulus can get the  !
! bus controller out of synchronization with the pod.  Two fault        !
! handlers trap pod timeout conditions that indicate the bus controller !
! is out of synchronization.  The recover() program is executed to      !
! resynchronize the bus controller and the pod.                         !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    recover     ()                        The 80286 microprocessor has a!
!                                          bus controller that is totally!
!                                          separate from the pod.  In    !
!                                          some cases the pod can get out!
!                                          of sync with the bus control- !
!                                          ler.  The recover program     !
!                                          resynchronizes the pod and the!
!                                          bus controller.               !
!                                                                       !
! GRAPHICS PROGRAMS CALLED:                                             !
!    (none)                                                             !
!                                                                       !
! Global Variables Modified:                                            !
!    recover_times                         Reset to Zero                !
!                                                                       !
! Local Variables Modified:                                             !
!    measure_dev                           Measurement device           !
!    stimulus_dev                          Stimulus device (overdrives) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

(continued on the next page)

Figure 4-140: Stimulus Program *(ready_6)*

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric recover_times
declare numeric done = 0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_timeout_enabled_line
   recover()
end handle
handle pod_timeout_recovered
   recover()
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

recover_times = 0

! Let GFI determine the measurement device.

   if (gfi control) = "yes" then
      measure_dev = gfi device
      measure_ref = gfi ref
   else
      print "Enter reference name of part to measure:"
      print "    (Chose U5 or U17)"
      measure_ref = "" \ input measure_ref
      measure_dev = clip ref measure_ref
   end if
   print "Stimulus Program READY_6"

! Set addressing mode and setup measurement device.

   podsetup 'enable ~ready' "off"
   podsetup 'standby function off'
   podsetup 'report power' "off"
   podsetup 'report forcing' "off"
   podsetup 'report intr' "off"
   podsetup 'report address' "off"
   podsetup 'report data' "off"
   podsetup 'report control' "off"
   setspace( getspace( "i/o", "byte" ))
   reset device measure_dev
   sync device measure_dev, mode "ext"
   enable device measure_dev, mode "high"
   edge device measure_dev, start "+", stop "count", clock "-"
   stopcount device measure_dev, count 4
```

(continued on the next page)

Figure 4-140: Stimulus Program *(ready_6) - continued*

```
! Prompt user to connect external lines.

   if measure_ref = "U17" then
      connect device measure_dev, start "U4-11", clock "U1-10", common "gnd"
   else
      connect device measure_dev, start "U17-9", clock "U1-10", common "gnd"
   end if

! External lines determine measurement.

   loop until done = 1
      arm device measure_dev
         read addr 0
         done = check_meas(measure_dev,"U4-11", "*", "U1-10", "*")
      readout device measure_dev
   end loop

   clearoutputs device measure_dev
   podsetup 'standby function on'
   podsetup 'enable ~ready' "on"

end program
```

Figure 4-140: Stimulus Program *(ready_6) - continued*

```
STIMULUS PROGRAM NAME: READY_6
DESCRIPTION:                                          SIZE:          70 BYTES

                    ------------------- Response Data --------------------
   Node       Learned                  Async Clk Counter                    Priority
Signal Src     With          SIG        LVL  LVL  Mode     Counter Range      Pin

U17-11         I/O MODULE               1 0     0 TRANS    0
```

Figure 4-141: Response File *(ready_6)*

Summary of Complete Solution for Ready Circuit 4.14.8.

The entire set of programs and files needed to test and GFI troubleshoot the Ready Circuit functional block is shown below. The format below is similar to a 9100A/9105A UUT directory (you could consider the functional block to be a small UUT), but in addition shows the use of each program and the location in this manual for each file.

UUT DIRECTORY
(Complete File Set for Ready Circuit)

Programs (PROGRAM):

TST_READY	Functional Test	Section 4.14.5
READY_1	Stimulus Program	Figure 4-130
READY_2	Stimulus Program	Figure 4-132
READY_3	Stimulus Program	Figure 4-134
READY_4	Stimulus Program	Figure 4-136
READY_5	Stimulus Program	Figure 4-138
READY_6	Stimulus Program	Figure 4-140

Stimulus Program Responses (RESPONSE):

READY_1	Figure 4-131
READY_2	Figure 4-133
READY_3	Figure 4-135
READY_4	Figure 4-137
READY_5	Figure 4-139
READY_6	Figure 4-141

Node List (NODE):

NODELIST	Appendix B

Text Files (TEXT):

Reference Designator List (REF):

REFLIST	Appendix A

Compiled Database (DATABASE):

GFIDATA	Compiled by the 9100A

OTHER FUNCTIONAL BLOCKS AND CIRCUITS 4.15.

The 9100A/9105A provides the capability to handle a number of special circuits or situations. Among these are watchdog timers forcing lines, feedback loops, and in-circuit component testing.

Watchdog Timers 4.15.1.

Watchdog timers usually interfere with testing and troubleshooting. If your UUT has a watchdog timer, your test procedure or program must disable it before performing tests.

Many watchdog timers initiate a master reset when they detect incorrect activity. Others may use a high-priority interrupt line to reset the system.

Whenever possible, physically disable the watchdog timer with a jumper or switch provided for that purpose. If the watchdog timer cannot be disabled at the UUT, the 9100A/9105A may be able to ignore it with the SETUP POD REPORT FORCING SIGNAL ACTIVE OFF keypad command, or disable it with a command like SETUP POD ENABLE READY ON/OFF. Be very careful, however, when doing this. Read the precautions about these commands in Section 4.15.2, "Forcing Lines."

Forcing Lines 4.15.2.

In some situations, forcing lines must be disabled (disconnected from the pod microprocessor) during a test. You can do this with the SETUP POD ENABLE READY ON/OFF keypad command ("READY" is a pod-dependent choice; some pods may call this line by a different name).

Exercise care whenever you disable a forcing line. Write or read commands to circuits that generate wait states through a Ready line may become unpredictable after the Ready line is disabled at the pod.

In addition to disabling forcing lines, you can also ignore them. The SETUP POD REPORT FORCING SIGNAL ACTIVE OFF keypad command will prevent the reporting of forcing lines. In this mode, the pod behaves normally but forcing conditions are not reported by the pod to the 9100A/9105A.

Exercise care with this mode also. The pod's hardware performance is not affected and the pod will continue reacting to the forcing line. If the UUT generates a permanent wait state using a forcing line, the pod will halt and the system will display a timeout message. Other fault-indicating signals on your UUT will also be ignored if the forcing line is disabled. Be sure that your UUT hardware is not affected by the same forcing line.

Breaking Feedback Loops 4.15.3.

Microprocessor-based systems often have several feedback loops. The microprocessor and the components tied to the data and address buses form a large feedback loop. Most of the loops in the system will be broken when the microprocessor is replaced by the pod, because the pod can selectively ignore or report conditions of status and forcing lines. However, there may be additional loops which are not broken by the pod.

Figure 4-125 shows a feedback loop in the Ready functional block of the Demo/Trainer UUT. The READY output (U1-4) is fed back as an input at U4-12.

To test a functional block that contains a feedback loop, drive all of its inputs, including the inputs connected to outputs that form the feedback loop, and measure the outputs. Use the I/O module to overdrive inputs while measuring signature, level, and count at the outputs.

Visual and Acoustic Interfaces 4.15.4.

Some circuits, such as LEDs and beepers, have both electrical characteristics and visual or acoustic characteristics . In general, stimulus programs should ignore the visual or acoustic

characteristics and measure only the electrical characteristics . The functional tests should prompt the test operator to verify the visual or acoustical characteristics .

If the functional test fails, use the *gfi test* command. If *gfi test* fails, start GFI troubleshooting. If the functional test fails and *gfi test* passes, the part is bad, since the part operates incorrectly but the electrical signals at the part are good.

In the case of the Parallel I/O functional block on the Demo/Trainer UUT, the functional test includes a prompt to the operator to verify the correct display on the LEDs. If the LEDs fail, the Parallel I/O functional test should perform a *gfi test,* which will run the stimulus programs and check the electrical properties. If *gfi test* passes (when the Parallel I/O functional test failed), it means that the electrical characteristics are good but the display is bad. The LEDs are bad and the operator should be prompted to replace them. If the *gfi test* fails, GFI troubleshooting can begin at the pin where the *gfi test* failed.

In-Circuit Component Tests 4.15.5.

If you wish, you can write TL/1 programs to test individual components rather than using the GFI to do so. These in-circuit component tests use a sequence of ones and zeroes defined with the TL/1 *storepatt* command and executed by the TL/1 *writepatt* command to overdrive the inputs of the component to be tested while measuring the signatures or level histories of its outputs. A test operator runs these tests by using the EXEC key to run the required program.

(This page is intentionally blank.)

Section 5
UUT Go/No-Go Functional Tests

PROGRAMMED GO/NO-GO FUNCTIONAL TESTING 5.1.

The UUT go/no-go test is the third of four modular levels in programming the 9100A, as shown in Figure 5-1. In this third level, the go/no-go test determines whether the UUT is good (passes) or bad (fails). The go/no-go test combines built-in functional test commands with functional tests designed by the programmer.

The go/no-go test is simple because it builds on the tests of functional blocks. It determines only whether the entire UUT is good or bad. It does not determine which functional block is causing a failure.

CREATING A PROGRAMMED GO/NO-GO FUNCTIONAL TEST 5.2.

Suppose a UUT has 14 functional blocks and a functional test is defined for each of them. One way to create a go/no-go test is to perform all 14 functional tests. Some blocks, however, can be tested indirectly by testing other blocks. For example, the bus buffer is assumed to be good if the ROM, RAM, and other blocks pass their tests. Therefore, a second way to create the go/no-go test is to perform functional tests only on functional blocks which cannot be tested indirectly by testing other blocks.

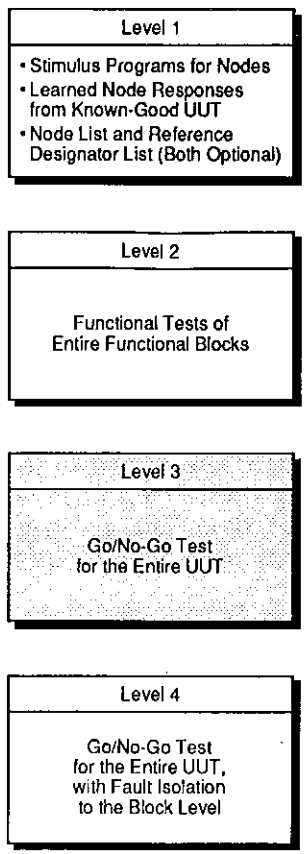


Figure 5-1: UUT Go/No-Go Functional Testing (Level 3)

Figure 5-2 shows the steps used to reach a go/no-go status decision. Care must be taken to ensure that your go/no-go test really does test the UUT for all possible faults.

Figure 5-3 shows the structure of a go/no-go functional test for the Demo/Trainer UUT. For this UUT, only six functional blocks need to be tested for the go/no-go functional test of the UUT: Microprocessor Bus, RAM, ROM, Parallel I/O, Serial I/O, and Video. The microprocessor bus test is run first because it is built-in, fast, and provides excellent diagnostic information. A failure on the microprocessor bus will cause most other circuits to fail, so it is most efficient to check this functional block first.

In the Demo/Trainer UUT, the following functional blocks are tested indirectly by the go/no-go test:

Clock and Reset
Ready Circuit
Interrupt Circuit
Bus Buffer
Dynamic RAM Timing
Address Decode
Video Control
Video RAM

Figure 5-4 is a listing of the go/no-go functional test program for the Demo/Trainer UUT. It calls the functional test for each of the functional blocks which must be tested directly for the UUT go/no-go functional test to be complete. The remaining functional blocks are tested indirectly; if they fail, one of the six blocks that is tested by the go/no-go test will fail also.

EVALUATING TEST EFFECTIVENESS 5.3.

The purpose of the go/no-go test is to determine whether the UUT is good or bad. Two measures are frequently used to evaluate how well a go/no-go functional test performs: node activity and fault coverage. Node activity is important because

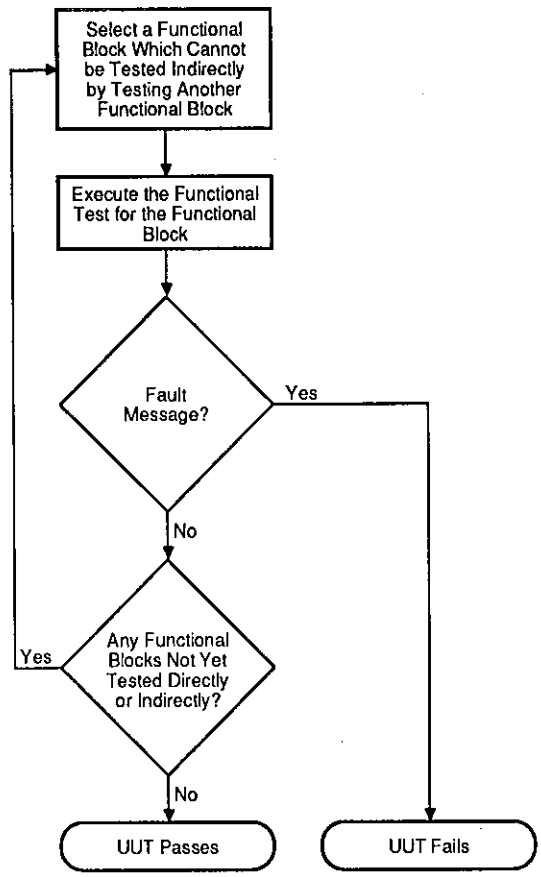


Figure 5-2: Go/No-Go Test Sequence

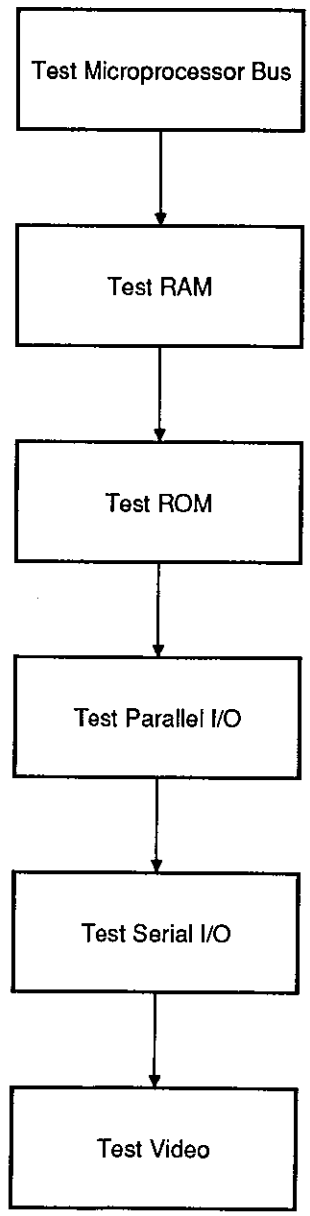


Figure 5-3: Demo/Trainer UUT Go/No-Go Test

```
program go_nogo

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The Go/No-Go program is the highest level of the functional testing   !
! and fault handlers.  The purpose of the Go/No-Go test is to determine !
! whether the UUT is good or bad.  This program executes six programs   !
! which test the six major functional blocks (Microprocessor Bus, ROM,  !
! RAM, Parallel I/O, Serial I/O, and Video functional blocks).          !
! By testing the six major functional blocks, the remaining             !
! functional blocks are indirectly tested.                              !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    test_bus    ()                      Test the microprocessor bus,   !
!                                        buffered bus, and address      !
!                                        select logic.                  !
!                                                                       !
!    test_rom    ()                      Test the ROM functional block  !
!                                        of the Demo/Trainer UUT.       !
!                                                                       !
!    test_ram    ()                      Test the RAM functional block  !
!                                        of the Demo/Trainer UUT.       !
!                                                                       !
!    test_pia    ()                      Test the PARALLEL I/O          !
!                                        functional block of the        !
!                                        Demo/Trainer UUT.              !
!                                                                       !
!    test_rs232  ()                      Test the SERIAL I/O            !
!                                        functional block and the       !
!                                        Interrupt Circuit functional   !
!                                        block of the Demo/Trainer UUT. !
!                                                                       !
!    test_video  ()                      Test the VIDEO circuit of the  !
!                                        Demo/Trainer UUT.              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! SETUP AND SYSTEM INITIALIZATION !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   podsetup 'report power' "on"      ! Turn on reporting functions except
   podsetup 'report intr' "off"      ! interrupt which is tested in the
   podsetup 'report address' "on"    ! SERIAL I/O test (test_rs232).
   podsetup 'report control' "on"
   podsetup 'report data' "on"
   podsetup 'report forcing' "on"

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! This is the go/no-go test which runs the major functional tests. !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   gfi clear                         ! CLEAR ALL GFI RECOMMENDATIONS
   connect clear "yes"               ! Clear all connect information.

   execute test_bus()
   execute test_rom()
   execute test_ram()
   execute test_pia()
   execute test_rs232()
   execute test_video()
end program
```

Figure 5-4: Go/No-Go Test for Demo/Trainer UUT

each node on the UUT must be exercised for a thorough functional test.

However, activity on each node is not a sufficient evaluation of test effectiveness. In addition, you need to evaluate how well your test detects faults in the UUT. This is done by injecting faults (such as stuck lows, stuck highs, intermittent highs, or intermittent lows) at each node in the UUT while running your functional test to see if the test fails. The 9100A/9105A probe (used as a source) provides a convenient tool for this purpose.

Fault coverage is the percentage of faults that will be detected by the functional test software. It is often measured as the ratio of the number of nodes where injected faults can be detected by a test to the total number of nodes in the UUT. This ratio is usually expressed in percent. If the fault coverage is not high, you can analyze the pattern of faults that are not detected to determine additions to your test program to increase the fault coverage.

EXECUTING UUT SELF-TESTS 5.4.

Self-test routines contained in UUT memory can be executed from the 9100A/9105A by pressing the RUN UUT key at the operator's keypad and entering the UUT's starting address of the routine. These self-test routines can also be run from TL/1 programs by using the *runuut* command. Self-test routines typically save their test results in UUT RAM. The 9100A/9105A can later read the appropriate RAM addresses to get these results.

An I/O module can generate one hardware breakpoint (system interrupt) upon detection of any user-defined combination of logic-highs and logic-lows on selected I/O module lines. This feature may be invoked at the operator's keypad (SET I/O MOD COMPARE WORD command), or through program execution. Once set up for a breakpoint, the I/O module continuously monitors the specified lines while other functions (such as RUN UUT) are performed. When the breakpoint event occurs, RUN UUT execution halts. A breakpoint message will interrupt any

current system activity. If a program is being executed, it may redirect the breakpoint message through a fault condition handler, as described in Section 6 of this manual.

A complete functional test for a UUT might begin with the BUS, RAM, and ROM tests, followed by execution of UUT self-test routines. By using RUN UUT breakpoints to detect addresses, data, and other UUT logic levels, the program can integrate the UUT's self-tests with 9100A/9105A functional tests.

Some pods can also generate UUT breakpoints without using the I/O module. For these pods, breakpoint-related softkeys appear when the RUN UUT key is pressed. Consult your pod manual for these pod-specific breakpoint capabilities, if any.

EXECUTING DOWNLOADED MACHINE CODE 5.5.

After part of the UUT RAM has been tested and found to be good, machine code can be downloaded to the tested RAM and executed. The machine code may be downloaded using a series of WRITE commands or the WRITE BLOCK command, which downloads an entire Motorola-format user file.

After the code is downloaded, you can execute it with the RUN UUT command, specifying the code's starting address. Although most testing can be done efficiently through the TL/1 test language, downloading machine code is useful when the code for a test already exists, when the testing must be done at machine-code speeds, or when a feature not supported by the pod must be used as part of the test.

The pod's microprocessor bus cycles are actually done at full UUT speed. The 9100A/9105A, however, is often slower than the UUT. For example, when the system performs a looping READ, each bus cycle is at full UUT speed but individual read operations are not done one immediately after the other.

Section 6
Identifying a Faulty Functional Block

After the go/no-go test determines that a UUT is faulty, the next step is to identify the failing functional block. Doing so before starting to troubleshoot will greatly improve troubleshooting efficiency because troubleshooting can begin *closer to the failure* and will take less time to reach the failing node. In addition, fault detection will be more accurate because the diagnostic test can check for special types of faults, such as bus contention, before troubleshooting begins.

Programs that identify faulty functional blocks are called diagnostic programs. Diagnostic programs, which are a subset of troubleshooting procedures, build on the UUT go/no-go test, functional tests of blocks, and stimulus programs. They are the last of the four modular levels in programming the 9100A, as shown in Figure 6-1. In this fourth programming level, fault condition handlers and *gfi hint* commands are added to the UUT go/no-go test to create a diagnostic program that traps faults and initiates tests of functional blocks that may be responsible for the fault, thereby isolating the block that is causing the UUT to fail. In addition, a failing output of the faulty block is identified as a starting point for backtracing toward the fault that causes the block to fail. At that point, GFI troubleshooting (the GFI key on the operator's keypad) can be used to backtrace to the bad node or component.

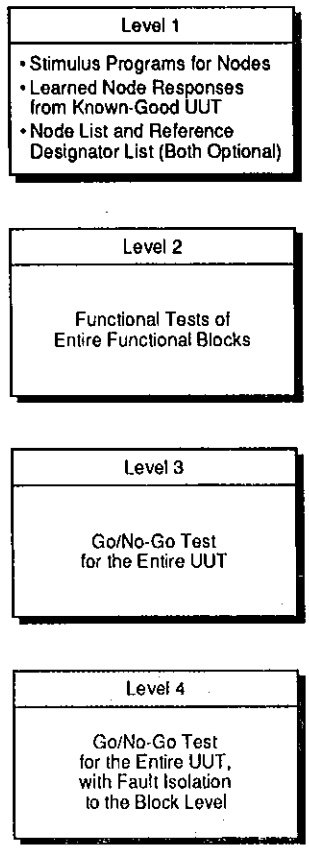


Figure 6-1: Diagnostic Programs (Level 4)

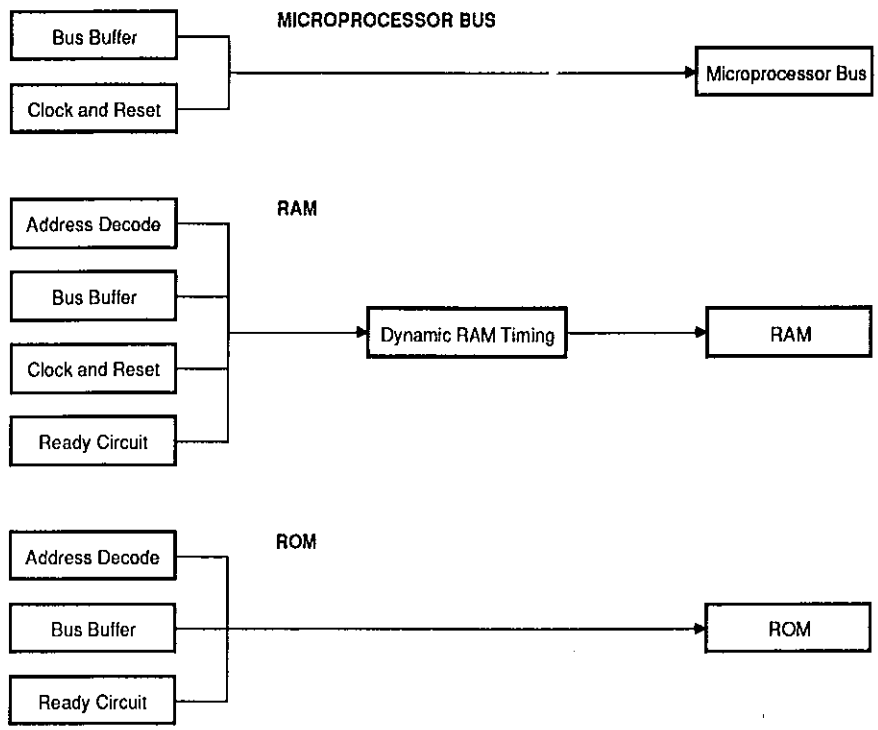
STRATEGY OF DIAGNOSTIC PROGRAMS 6.1.

The first step in developing a diagnostic strategy is to draw a diagram showing the major functional blocks used in the go/no-go functional test. Next, show all other functional blocks that provide input to these major functional blocks. Figure 6-2 shows such a diagram for the Demo/Trainer UUT. The figure shows six sets of functional blocks, one for each major functional block tested by the go/no-go functional test. The blocks on the left provide input to the blocks on the right, and the blocks tested by the go/no-go functional test are on the right side of each set.

The task of the diagnostic program is to select a failing functional block for troubleshooting and to generate an appropriate starting point (or points) where GFI can begin automated troubleshooting. When a major functional block fails, you know that one or more outputs of the block are bad. But it doesn't necessarily mean that the block itself is bad; bad inputs to the major functional block may be causing the block to fail. How do you continue from there to isolate the failing block and select an efficient starting point for GFI?

One diagnostic strategy is to test blocks that provide input to the failing major block. Isolating the block causing a failure involves tracing from the right-hand side toward the left, testing each block in the path until one is found with good inputs and bad outputs. This strategy works best when the string of blocks leading up to a major block is short. Such is the case for most of the sets of blocks in Figure 6-2.

A second diagnostic strategy, helpful when you have a longer string of blocks leading up to a failing major block, is to divide the blocks in half and begin testing a block halfway between the first block in the string and the major block at the end. If the middle block passes, keep dividing the failing string of blocks in half and testing a middle block. If the middle block fails, test the blocks to the left starting at the middle block. This second strategy would be appropriate for the Video set of blocks in Figure 6-2.



(continued on the next page)

Figure 6-2: Inputs to Functional Blocks

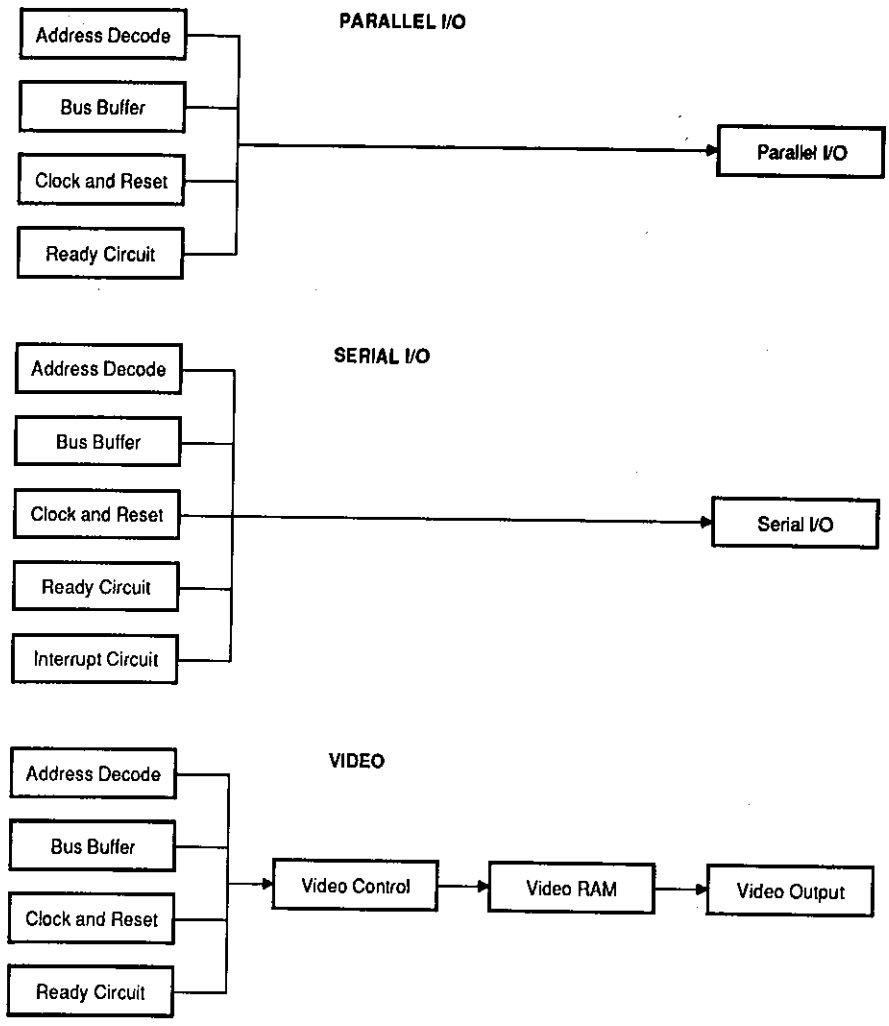


Figure 6-2: Inputs to Functional Blocks- *continued*

Another strategy, used when a fault is likely to be near a failing output pin of the failing major block, is to begin GFI backtracing directly from the failing output pin, without checking the inputs to the major functional block.

Diagnostic programs can speed up troubleshooting by starting GFI closer to the actual problem. On the other hand, isolating the failure to a very small area may require more time than is saved in reduced troubleshooting time. There is a balance between isolating the failure to a very small area and doing no isolation of the failing circuit. Decisions on when to start GFI and when to isolate the failure to a smaller area depend on your UUT and the relative cost of additional programming effort compared to the resulting savings in troubleshooting time.

IMPLEMENTING THE STRATEGY FOR DIAGNOSTIC PROGRAMS 6.2.

Figure 6-3 shows a typical process to implement a diagnostic program strategy. The diagnostic program executes a functional test for each major functional block. If a fault condition is generated during the test, the major functional block is possibly faulty. To verify this suspicion, the inputs to the functional block are checked. If the inputs are all good, then the major functional block is indeed faulty. However, if one of the inputs to a major functional block is not good, the fault probably lies in the functional blocks which provide input to the major functional block. In this case, the input functional blocks become the suspect blocks and their inputs are checked. This process continues until a block is found with all good inputs but a bad output.

When this faulty functional block is identified, appropriate GFI hints are generated to indicate the node (or nodes) where GFI should start troubleshooting.

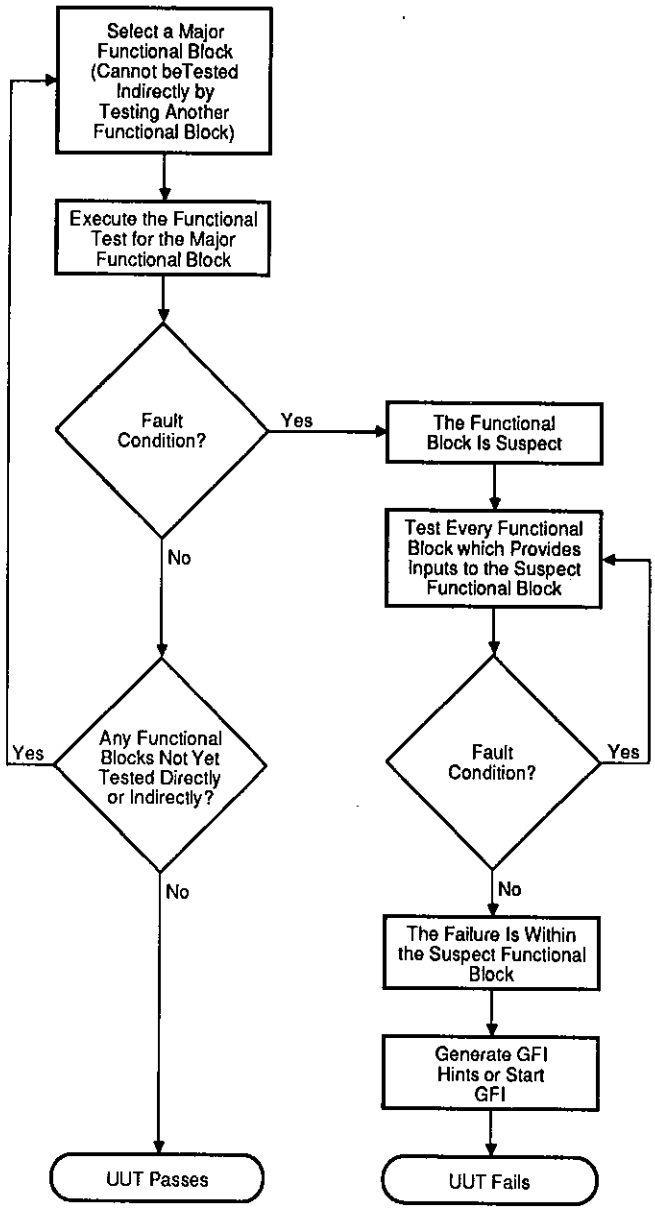


Figure 6-3: Identifying a Faulty Functional Block

DIAGNOSIS USING FAULT CONDITION HANDLERS 6.3.

Fault condition handlers provide the means for communicating 9100A/9105A functional test failure information to the operator for keystroke troubleshooting or to GFI for automated troubleshooting.

What are Fault Condition Handlers? 6.3.1.

A fault condition is generated or "raised" in one of two ways:

- A built-in TL/1 function is run, and the UUT does not respond correctly. For example, a microprocessor address line cannot be driven to logic-high during a read or write operation.
- A *fault* command is executed in a TL/1 program.

A fault condition handler is a TL/1 procedure, called by a fault condition of the same name, that responds in some way to the fault condition. For example, the handler might try to determine the cause of the fault.

Each fault condition has a name. Fault conditions created by built-in functions have defined names and parameters, listed in *TL/1 Reference Manual* appendices. Fault conditions created by your *fault* commands may have any name, including the same name used by the built-in functions.

When a fault condition is raised, the system halts execution of the current program. If your program contains a fault condition handler with the same name as the fault condition, the program statements inside the handler are executed. After the handler is finished, execution of your program resumes where it left off.

If your program does not contain an appropriate fault condition handler, execution of the program terminates and its calling program (if any) is searched for a fault condition handler with

the specified fault condition name. This process continues until an appropriate handler is found. If no handler is found, a fault message will appear on the operator's display.

For more information on fault condition handlers, see Section 3.7 of the *Programmer's Manual.*

Using Fault Condition Handlers 6.3.2.

The UUT go/no-go test should test only those functional blocks that cannot be tested indirectly by other blocks. When the go/no-go test detects a failure, the diagnostic program is used to identify the failing block and to identify a failing node as a starting point for troubleshooting.

To use fault condition handlers in a diagnostic program, you need to do two programming tasks for each handler:

1. Use the *fault* command (with an appropriate fault condition that you create) to generate the fault condition if a test (or part of a test) of a functional block fails. For example, if the diagnostic program finds that the functional test of the video output circuitry fails, you might choose to generate a fault condition named *video_output.*

2. Create a handler for this fault condition. The handler should check other input blocks to isolate the failing functional block. It might also do further testing to narrow down the zone of failure within a failing functional block. And the handler will generate the appropriate starting point for GFI by using the *gfi hint* command.

A Diagnostic Test Example 6.3.3.

Suppose the video circuitry is failing. Testing begins with execution of the *go/no-go2* program, listed in Section 6.4 of this manual. This program has many fault condition handlers at the

beginning, and it has six *execute* statements at the end that actually execute the go/no-go test. Each of these *execute* statements executes a different functional test program for a major functional block. And each of these functional test programs include the necessary fault condition handlers to generate GFI hints appropriate for the fault condition encountered (a listing for each of these programs is contained in Section 6.5 of this manual). The GFI hints are very important to the troubleshooting process; they are the means by which the 9100A/9105A communicates the results of its functional testing to provide efficient starting points for GFI troubleshooting.

Suppose that the failing video circuitry does not affect any of the six major functional blocks except *test_video2*. In this case, *test_bus2*, *test_rom2*, *test_ram2*, *test_pia2*, and *test_rs232b* all pass, but *test_video2* fails. The *test_video2* test is really the test of the Video Output functional block. If this test fails, a video fault condition is generated (suppose the *video_scan* fault condition is generated). Since the *test_video2* program has a handler for *video_scan*, the program statements inside this handler are executed.

Once the hints to GFI are passed, execution of the video fault condition handler *(video_scan)* ends, the test program *(test_video2)* ends, and the diagnostic program *(go_nogo2)* ends. A message appears on the operator's display saying that GFI hints have been generated, and that GFI should be run.

The diagnostic program is structured so that only one failure is isolated at a time. The problem should be isolated with GFI and fixed when it is detected. It is appropriate to repair an isolated fault before testing any further, since apparent multiple failures often result from one physical problem on a board. For example, a short between two nodes can appear as two failures. After a fault has been repaired, the diagnostic program should be run again to find other faults or to verify that no more faults can be found.

DIAGNOSTIC PROGRAM FOR THE DEMO/TRAINER UUT

6.4.

```
program go_nogo2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The Go/No-go program is the highest level of the functional testing  !
! and fault condition handlers. The purpose of the Go/No-go test is to !
! determine whether the UUT is good or bad. This program executes six  !
! programs which test the six major functional  blocks (Microprocessor !
! Bus, ROM, RAM, Parallel I/O, Serial I/O, and Video). By testing the  !
! six major functional blocks, the remaining functional blocks are     !
! indirectly tested.                                                   !
!                                                                      !
! If the Go/No-go test detects a faulty UUT, further fault isolation is !
! performed to isolate which circuit is causing the failure. The fault !
! condition handlers in the Go/No-go program and the other functional  !
! test programs perform the fault isolation. The fault condition       !
! handlers included in this program are handlers for those fault       !
! conditions which may occur during any of the six major functional    !
! tests.                                                               !
!                                                                      !
! The major functional test programs include fault condition handlers  !
! for fault  conditions which are only generated within that program.  !
! The first three programs (TEST_BUS, TEST_ROM, and TEST_RAM) use      !
! built-in TL/1 tests and the built-in fault condition handlers that   !
! are documented in the 9100/9105A TL/1 Reference Manual.              !
!                                                                      !
!                                                                      !
! TEST PROGRAMS CALLED:                                                !
!    test_bus2                           Test the microprocessor bus,  !
!                                        buffered bus, and address     !
!                                        select logic.                 !
!                                                                      !
!    test_rom2                           Test the ROM functional block !
!                                        of the Demo/Trainer UUT.      !
!                                                                      !
!    test_ram2                           Test the RAM functional block !
!                                        of the Demo/Trainer UUT.      !
!                                                                      !
!    test_pia2                           Test the PARALLEL I/O         !
!                                        functional block of the       !
!                                        Demo/Trainer UUT.             !
!                                                                      !
!    test_rs232b                         Test the SERIAL I/O functional!
!                                        block and the Interrupt       !
!                                        Circuit functional block of   !
!                                        the Demo/Trainer UUT.         !
!                                                                      !
!    test_video2                         Test the VIDEO circuit of the !
!                                        Demo/Trainer UUT.             !
!                                                                      !
!    recover                             The 80286 microprocessor has a!
!                                        bus controller that is totally!
!                                        separate from the pod.  In    !
!                                        some cases, the pod can get   !
!                                        out of sync with the bus      !
!                                        controller.  The recover      !
!                                        program resynchronizes the pod!
!                                        and the bus controller.       !
!                                                                      !
```

```
! FUNCTIONS CALLED:                                                    !
!    retry_access (access, addr, control) This function is executed when!
!                                         a pod_timeout_recovered fault !
!                                         condition occurs.  This       !
!                                         function repeats the attempted!
!                                         access that failed and        !
!                                         determines if the access can  !
!                                         be sucessfully repeated.      !
!                                                                       !
! Global Variables Modified:                                            !
!    recover_times                        Reset to Zero                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare
   global numeric recover_times          ! Count of executing recover().
end declare

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! GENERAL PURPOSE FAULT CONDITION HANDLERS                              !
!                                                                       !
! The built-in fault conditions "pod_addr_tied", "pod_ctl_tied",        !
! "pod_data_incorrect" and pod_data_tied are generated when the pod     !
! detects a stuck or tied line at the pod socket.  These fault          !
! conditions are not handled because the diagnostic message for these   !
! faults cannot be made better by additional testing.  If one of these  !
! fault conditions occurs, the built-in fault message will be displayed!
! and the UUT needs to be repaired.                                     !
!                                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle pod_forcing_active (mask)
   declare string mask
   declare global numeric tlo
   declare string clear_screen = "\1B[2J"
   print on tlo ,clear_screen, "POD Forcing Lines Active fault"
   fault forcing_lines mask mask              ! Redirect fault
end handle

handle pod_interrupt_active (mask)
   declare string mask
   declare global numeric tlo
   declare string clear_screen = "\1B[2J"
   print on tlo ,clear_screen, "POD Interrupt Line Active fault"

! Get the last two characters of the 64 bit string mask and decode to INTR/NMI

   lines = val(mid(mask, len(mask)-3, 2),16)
   if (lines and $10) <> 0 then
      execute tst_intrpt()
   else if (lines and 1) <> 0 then
      fault NMI_active
   end if
end handle

handle pod_misc_fault
   fault bad_power                        ! Redirect fault
end handle
```

```
handle pod_special
end handle

handle pod_timeout_bad_pwr
   declare global numeric tlo
   declare string clear_screen = "\1B[2J"
   print on tlo ,clear_screen, "POD timeout bad power fault"
   fault bad_power                                  ! Redirect fault
end handle

handle pod_timeout_enabled_line (mask)
   declare string mask
   declare global numeric tlo
   declare string clear_screen = "\1B[2J"
   print on tlo ,clear_screen, "POD Timeout Enabled line fault"
   fault forcing_lines mask mask                    ! Redirect fault
end handle

handle pod_timeout_no_clk
   declare global numeric tlo
   declare string clear_screen = "\1B[2J"
   print on tlo ,clear_screen, "POD Timeout No Clock at POD Pin 31"
   execute tst_clock()                              ! Test Clock and Reset
end handle

handle pod_timeout_recovered (access_attempted, ctl, addr)
   declare string access_attempted
   declare numeric ctl = $E0000000
   declare numeric addr = $E0000000
   declare global numeric tlo
   declare string clear_screen = "\1B[2J"
   declare global numeric repeated_timeouts

   print on tlo ,clear_screen, "pod timeout recovered:  "
   podsetup 'enable ~ready' "off"
   podsetup 'enable hold' "off"
   podsetup 'report forcing' "off"
   repeated_timeouts = repeated_timeouts + 1

! DISABLE all lines that can be enabled, retry access, then turn enable
! lines on until the access cannot be repeated.  The lines that can be
! enabled on the 80286 are Hold and Ready.

   if repeated_timeouts > 10 then
      fault dead_kernel
   else if retry_access(access_attempted, ctl, addr) fails then
      fault dead_kernel
   else
      podsetup 'enable hold' "on"
      if retry_access(access_attempted, ctl, addr) fails then
         fault hold_circuit
      else
         podsetup 'enable ~ready' "on"
         if retry_access(access_attempted, ctl, addr) fails then
            execute tst_decode()
            execute tst_ready()
         else
            print on tlo ,clear_screen
         end if
      end if
   end if
end handle
```

```
handle pod_timeout_setup
end handle

handle pod_uut_power
   fault bad_power                                  ! Redirect fault
end handle

handle iomod_dce
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Redirected Fault Handlers !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle forcing_lines (mask)
   declare string mask
   declare global numeric recover_times

! attempt to recover synchronization between pod and bus controller before
! testing the decode, ready or clock circuits.  If the recover procedure
! has been executed at least twice, then go ahead and test decode, ready or
! the clock circuit.

   if recover_times < 2 then
      execute recover()
   else
      lines = val(mid(mask, len(mask)-7, 8),16)
      if (lines and 1) <> 0 then
         execute tst_decode()
         execute tst_ready()
      else if (lines and $10) <> 0 then
         execute tst_clock()                        ! Test Clock and Reset
      end if

! The status lines HOLD, PEREQ, BUSY and ERROR are not used in the
! Demo/Trainer UUT.  Display a message if one of these lines is active
! and wait for the condition to be fixed.

      loop while (lines and $E2) <> 0
         print on tlo ,clear_screen
         if (lines and 2) <> 0 then
            print on tlo ,"HOLD is active; Press RESET to continue"
         else if (lines and $20) <> 0 then
            print on tlo ,"PEREQ is active; Press RESET to continue"
         else if (lines and $40) <> 0 then
            print on tlo ,"~BUSY is active; Press RESET to continue"
         else if (lines and $80) <> 0 then
            print on tlo ,"~ERROR is active; Press RESET to continue"
         end if
         wait time 2000
      end loop
   end if
end handle
```

```
handle bad_power
   declare global numeric t2o
   declare string clear_screen = "\1B[2J"
   declare global string messg
   print on t2o ,messg+"FAULT  DETECTED"
   loop until (readstatus() and $3D00) = 0
      fail($14)
      if (readstatus() and $3C00) = $3C00 then
         print on tlo ,clear_screen, "POD UUT Power"
         print on tlo ,"POWER_UP and press RESET on Trainer UUT"
         wait time 2000
         print on tlo ,clear_screen, "CONTINUING..."
      else
         if (readstatus() and $100) <> 0 then fault 'CAP failure at POD Pin 52'
         if (readstatus() and $400) <> 0 then fault 'POWER failure at POD Pin 30'
         if (readstatus() and $800) <> 0 then fault 'POWER failure at POD Pin 62'
         if (readstatus() and $1000) <> 0 then fault 'GROUND failure at POD Pin 35'
         if (readstatus() and $2000) <> 0 then fault 'GROUND failure at POD Pin 9'
      end if
   end loop
   untested($14)
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Functions                                                                !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

function retry_access(ACCESS, ADDR, CTL)

! Retry last access performed using parameters from fault handlers.

   handle pod_timeout_bad_pwr
      fault
   end handle

   handle pod_timeout_enabled_line
      fault
   end handle

   handle pod_timeout_no_clk
      fault
   end handle

   handle pod_timeout_recovered
      fault
   end handle

   handle pod_timeout_setup
      fault
   end handle
```

```
   declare string ACCESS
   declare numeric CTL
   declare numeric ADDR

   if ADDR <> $E0000000 then
      address = ADDR
   else if CTL <> $E0000000 then
      address = CTL
   else
      address = 0
   end if
   if ACCESS = "READ" then
      if read addr address fails then fault
   else if ACCESS = "WRITE" then
      if write addr address, data $A5C3 fails then fault
   end if

end function

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! SETUP AND SYSTEM INITIALIZATION !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   recover_times = 0
   execute recover()                    ! Recover synchronization between POD
                                        ! and the 80288 bus controller.

   podsetup 'report power' "on"         ! Turn on reporting functions except
   podsetup 'report intr' "off"         ! interrupts which is tested in the
   podsetup 'report address' "on"       ! SERIAL I/O test (test_rs232b).
   podsetup 'report control' "on"
   podsetup 'report data' "on"
   podsetup 'report forcing' "on"

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! This is the go/no-go test which runs the major functional tests. !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   gfi clear                            ! CLEAR ALL GFI RECOMENDATIONS
   connect clear "yes"                  ! Clear all connect information.

   execute test_bus2 ()
   execute test_rom2 ()
   execute test_ram2 ()
   execute test_pia2 ()
   execute test_rs232b ()
   execute test_video2 ()

end program
```

FUNCTIONAL BLOCK TESTS FOR THE DEMO/TRAINER UUT DIAGNOSTIC PROGRAM 6.5.

This section contains the following functional test programs, which are necessary to support the diagnostic program for the Demo/Trainer UUT:

test_bus2	Tests the Microprocessor Bus functional block.
test_pia2	Tests the Parallel I/O function block.
test_ram2	Test the RAM functional block.
test_rom2	Tests the ROM function block.
test_rs232b	Tests the Serial I/O function block.
test_video2	Tests the video circuitry (the Video Control, Video RAM, and Video Output functional blocks).

These programs are much like the programs by the same name found in Section 4 and used in Section 5 of this manual. However, these programs also contain the necessary fault condition handlers and *gfi hint* commands to tell GFI where to start backtracing if the functional block fails.

```
program test_bus2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the Microprocessor Bus.                            !
!                                                                       !
! This program tests the unbuffered microprocessor bus, performs an     !
! access at each decoded address of the buffered bus, and checks the    !
! data bus for bus contention (where a component outputs onto the data  !
! bus at incorrect times).   If bus contention is detected then the     !
! program TST_CONTEN is executed.  TST_CONTEN checks for incorrect      !
! enable line conditions on all the components on the buffered data bus.!
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    tst_conten (addr, data_bits)        Test for bus contention on     !
!                                        the data bus by checking the   !
!                                        enable lines of all devices    !
!                                        on the data bus.               !
!                                                                       !
! Local Constants:                                                      !
!    ZERO_AT_ROM0                        Address of zero data in ROM0   !
!    ZERO_AT_ROM1                        Address of zero data in ROM1   !
!    IO_BYTE                             I/O BYTE address specifier     !
!    MEM_WORD                            MEMORY WORD address specifier  !
!                                                                       !
! Local Variables Modified:                                             !
!    x                                   value returned from a read     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  declare numeric ZERO_AT_ROM0 = $E002A ! Location in ROM0 where 0 exists
  declare numeric ZERO_AT_ROM1 = $F0022 ! Location in ROM1 where 0 exists

! Setup Statements

   podsetup 'enable ~ready' "on"
   podsetup 'report forcing' "on"
   IO_BYTE = getspace space "i/o", size "byte"
   MEM_WORD = getspace space "memory", size "word"

! Test the Unbuffered Microprocessor Bus.

   testbus addr 0

! Test the Extended Microprocessor Bus and Address Decoding.

   setspace (MEM_WORD)
   read addr 0                  ! RAM BANK 0
   read addr $10000             ! RAM BANK 1
   write addr $20000, data 0    ! VIDEO RAM (write only)
   read addr $30000             ! INTERRUPT POLL
   read addr $E0000             ! ROM BANK 0
   read addr $F0000             ! ROM BANK 1
   setspace (IO_BYTE)
   read addr 0                  ! VIDEO SELECT
   read addr $2000              ! RS232 SELECT
   read addr $4000              ! PIA SELECT
```

```
! Test for Bus Contention driving lines low by accessing unused address space

   setspace (MEM_WORD)
   x = read addr $50000                          ! SPARE-2 ADDRESS SPACE
   if x <> $FFFF then
      execute tst_conten( $50000, cpl(x) and $FFFF)
      return
   end if

! Test for Bus Contention driving lines high by reading and writing RAM
! If failure then check for bad RAM by reading zeros from 2 other devices.

   write addr 0, data 0                          ! WRITE and READ RAM addr 0
   x = read addr 0                               ! If fails then check for bad RAM
   if x <> 0 then                                ! by reading 0's at ROM0 and ROM1
      if (read addr ZERO_AT_ROM0) <> 0 then
         if (read addr ZERO_AT_ROM1) <> 0 then
            execute tst_conten( 0, x)
            return
         end if
      end if
   end if

end program
```

```
program test_pia2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the PARALLEL I/O functional block.                 !
!                                                                       !
! This program tests the PARALLEL I/O functional block of the          !
! Demo/Trainer.  The two LEDs and the four pushbutton switches are      !
! tested.  The test operator is prompted to visually inspect the LEDs   !
! as the LEDs count a series of numbers.                                !
!                                                                       !
! TEST PROGRAMS CALLED:                                                 !
!    abort_test (ref-pin)                   If gfi has an accusation,   !
!                                           display the accusation;     !
!                                           otherwise create a gfi hint !
!                                           for the ref-pin and terminate !
!                                           the test program (GFI begins !
!                                           troubleshooting).           !
!                                                                       !
! TEST FUNCTIONS CALLED:                                                !
!    keys        (key_number)               Test Demo/Trainer pushbutton !
!                                           key key_number.  Prompt test !
!                                           operator to push the key.   !
!                                                                       !
!    leds        (led_addr, led_name)       Test Demo/Trainer LED led_name!
!                                           which is driven by the PIA and!
!                                           has the address led_addr.   !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare global numeric tlb                 ! Terml buffered output & input
declare global numeric tli                 ! Terml unbuffered input

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT CONDITION HANDLERS:                                           !
!      These fault conditions are generated by the this program.  These !
!      handlers perform isolation of the faulty circuit.  The handlers  !
!      which isolate the LED problems perform a GFI test on the LED.    !
!      If all signals are good and the test operator has failed the LED,!
!      then the LED is accused as a bad component.                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle 'PIA LED A failed'
   declare global string rev
   declare string newline = "\nl"

   if gfi test "U32-1" fails then
      abort_test("U32-1")
   else
      if gfi test "U33-1" fails then
         abort_test("U33-1")
      else if gfi test "U33-13" fails then
         abort_test("U33-13")
      else if gfi test "U33-10" fails then
         abort_test("U33-10")
      else if gfi test "U33-8" fails then
         abort_test("U33-8")
      else if gfi test "U33-7" fails then
         abort_test("U33-7")
      else if gfi test "U33-2" fails then
         abort_test("U33-2")
```

```
        else if gfi test "U33-11" fails then
           abort_test("U33-11")
        else if gfi test "U33-6" fails then
           abort_test("U33-6")
        else
           print rev, newline,"LED A IS BAD", newline, "REPLACE LED A"
        end if
     end if
  end handle

  handle 'PIA LED B failed'
     declare global string rev
     declare string newline = "\nl"

     if gfi test "U46-1" fails then
        abort_test("U46-1")
     else
        if gfi test "U47-1" fails then
           abort_test("U47-1")
        else if gfi test "U47-13" fails then
           abort_test("U47-13")
        else if gfi test "U47-10" fails then
           abort_test("U47-10")
        else if gfi test "U47-8" fails then
           abort_test("U47-8")
        else if gfi test "U47-7" fails then
           abort_test("U47-7")
        else if gfi test "U47-2" fails then
           abort_test("U47-2")
        else if gfi test "U47-11" fails then
           abort_test("U47-11")
        else if gfi test "U47-6" fails then
           abort_test("U47-6")
        else
           print rev, newline, "LED B IS BAD", newline, "REPLACE LED B"
        end if
     end if
  end handle

  handle 'PIA KEY 1 failed'
     abort_test("U31-14")
  end handle

  handle 'PIA KEY 2 failed'
     abort_test("U31-15")
  end handle

  handle 'PIA KEY 3 failed'
     abort_test("U31-16")
  end handle

  handle 'PIA KEY 4 failed'
     abort_test("U31-17")
  end handle
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Functions                                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

function keys(keynum)
   declare numeric keynum                    ! Number of key to test.
   declare string norm = "\1B[0m"            ! Normal video escape string
   declare string rev  = "\1B[0;7m"          ! Reverse video escape string
   declare string entry
   declare string fail = ""
   declare global numeric tlb
   declare global numeric tli

   mask = setbit(keynum - 1)

   loop until fail = chr($D)                             ! loop until YES key
      print on tlb ,"\nlPress ", rev," UUT KEY ", keynum," ",norm," pushbutton"
      print on tlb ,"Press any 9100 key if test is stuck"
      loop until (poll channel tli, event "input") = 1
         if ((read addr $4004) and mask) = 0 then return
      end loop
      loop until (poll channel tli, event "input") = 0    ! Flush input buffer
         input on tli ,entry
      end loop
      print on tlb ,"\nlPress ",rev," YES ",norm," to fail KEY ",keynum," test,"
      print on tlb ,"Press "+rev+" NO  "+norm+" to continue key test,"
      input on tli ,fail
   end loop
   print on tlb ,"\nl\nl"
   fault                                     ! Fail Key test (set termination
end function                                 ! status of function to fail.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

function leds(led_addr, led_name)
   declare numeric led_addr
   declare string led_name
   declare string key
   declare string norm = "\1B[0m"
   declare string bold = "\1B[1m"
   declare string rev = "\1B[7m"
   declare string clear_screen = "\1B[2J"
   declare string no_auto_linefeed = "\1B[20h"
   declare global numeric tli
   declare numeric array [0:10] numbers

   numbers [0] = $C0    \    numbers [5] = $92
   numbers [1] = $F9    \    numbers [6] = $82
   numbers [2] = $A4    \    numbers [7] = $F8
   numbers [3] = $B0    \    numbers [8] = $80
   numbers [4] = $99    \    numbers [9] = $98
   NO = chr($7F)        \    YES = chr( $D)

   print norm, clear_screen, "Watch LED ", led_name, " count"
   print "Press ", rev, " ENTER ", norm, " key to start LED counting."
   input  key
   print clear_screen

   for i = 0 to 9
      write addr led_addr, data numbers [i]
      wait time 500
   next
```

```
    write addr led_addr, data $7F
    print  clear_screen, "\1B[201"
    print  "\1B[1;1fDid LED ", led_name, " display ALL segments off, then"
    print  "\1B[2;1fdigits 0 to 9, then only the Decimal Point ?"
    print  "\1B[3;fpress: "+rev+" YES "+norm+" or "+rev+" NO  "+norm
    loop until key = YES or key = NO
       input on tli ,key
       if key = NO then fault
    end loop
    write addr led_addr, data $FF \  print  no_auto_linefeed,clear_screen

end function

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! PARALLEL I/O Test.                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    tlb = open device "/term1", as "update", mode "buffered"
    tli = open device "/term1", as "input", mode "unbuffered"
    execute pia_init()

    if leds($4000, "A") fails then fault 'PIA LED A failed' \ return
    if leds($4002, "B") fails then fault 'PIA LED B failed' \ return

    if keys(1)  fails then fault 'PIA KEY 1 failed' \ return
    if keys(2)  fails then fault 'PIA KEY 2 failed' \ return
    if keys(3)  fails then fault 'PIA KEY 3 failed' \ return
    if keys(4)  fails then fault 'PIA KEY 4 failed' \ return

end program
```

```
program test_ram2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the RAM functional block.                           !
!                                                                        !
! This program tests the RAM functional block of the Demo/Trainer.  The !
! TL/1 testramfast command is used to test the RAMs.  If the RAMs are    !
! found to be faulty, then one of twelve built-in fault conditions is    !
! generated.                                                             !
!                                                                        !
! TEST PROGRAMS CALLED:                                                  !
!    abort_test  (ref-pin)               If gfi has an accusation,       !
!                                        display the accusation;         !
!                                        otherwise create a gfi hint     !
!                                        for the ref-pin and terminate   !
!                                        the test program (GFI begins    !
!                                        troubleshooting).               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT CONDITION HANDLERS:                                            !
!      Built-in testramfast fault condition handlers                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle ram_addr_fault (data_mask)
   declare numeric data_mask
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRAM addr line fault detected,  CONTINUING"
   fault ram_component data_bits data_mask
end handle

handle ram_addr_addr_tied (data_mask)
   declare numeric data_mask
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRAM addr lines tied detected,  CONTINUING"
   fault ram_component data_bits data_mask
end handle

handle ram_addr_data_tied (data_expected, data)
   declare numeric data_expected
   declare numeric data
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRAM addr-data tied detected,  CONTINUING"
   fault ram_component data_bits (data xor data_expected)
end handle

handle ram_addr_data_tied_unconfirmed (data_expected, data)
   declare numeric data_expected
   declare numeric data
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRAM addr-data tied detected,  CONTINUING"
   fault ram_component data_bits (data xor data_expected)
end handle
```

```
handle ram_data_data_tied (data_expected, data)
   declare numeric data_expected
   declare numeric data
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\n\nRAM data lines tied detected,  CONTINUING"
   fault ram_component data_bits (data xor data_expected)
end handle

handle ram_data_fault (data)
   declare numeric data
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\n\nRAM data line fault detected,  CONTINUING"
   fault ram_component data_bits data
end handle

handle ram_data_incorrect (data_expected, data)
   declare numeric data_expected
   declare numeric data
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\n\nBAD RAM data detected,  CONTINUING"
   fault ram_component data_bits (data xor data_expected)
end handle

handle ram_data_high_tied (data_expected, data)
   declare numeric data_expected
   declare numeric data
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\n\nRAM data tied high detected,  CONTINUING"
   fault ram_component data_bits (data xor data_expected)
end handle

handle ram_data_low_tied (data_expected, data)
   declare numeric data_expected
   declare numeric data
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\n\nRAM data tied low detected,  CONTINUING"
   fault ram_component data_bits (data xor data_expected)
end handle

handle ram_cell_cell_tied (data_expected, data)
   declare numeric data_expected
   declare numeric data
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\n\nRAM cells tied detected,  CONTINUING"
   fault ram_component data_bits (data xor data_expected)
end handle

handle ram_cell_low_tied (data_expected, data)
   declare numeric data_expected
   declare numeric data
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\n\nRAM cell tied low detected,  CONTINUING"
   fault ram_component data_bits (data xor data_expected)
end handle
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Redirected fault handler                                               !
!                                                                          !
!   The RAM block can fail if a problem exists with the ready circuit.    !
!   So test the ready circuit, then if the ready circuit is good, use     !
!   the data bits parameter passed from the testramfast built-in fault    !
!   handlers to test the failing RAM IC.  If the RAM IC is good then      !
!   test the data bus at the bus buffers.  (Testing the data bus buffer   !
!   will detect any problem in the data bus).                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle ram_component (data_bits)
   declare numeric data_bits
   declare string array [0:$15] ram_ic

   ram_ic[0] = "U55"    \  ram_ic[1] = "U54"              ! RAMs U55, U54
   ram_ic[2] = "U53"    \  ram_ic[3] = "U52"              ! RAMs U53, U52
   ram_ic[4] = "U51"    \  ram_ic[5] = "U50"              ! RAMs U51, U50
   ram_ic[6] = "U49"    \  ram_ic[7] = "U48"              ! RAMs U49, U48
   ram_ic[8] = "U41"    \  ram_ic[9] = "U40"              ! RAMs U41, U40
   ram_ic[10] = "U39"  \  ram_ic[11] = "U38"              ! RAMs U39, U38
   ram_ic[12] = "U37"  \  ram_ic[13] = "U36"              ! RAMs U37, U36
   ram_ic[14] = "U35"  \  ram_ic[15] = "U34"              ! RAMs U35, U34

! If ready circuit is untested, then check Ready circuit

   if(gfi status "U1-4") = "untested" then
      if gfi test "U1-4"  fails then abort_test("U1-4" )
   end if

! Check highest order ram that is failing, using ram_ic array to get refname.

   if data_bits <> 0 then
      bad_ram_ref = ram_ic[msb(data_bits)] + "-1"
      if gfi test bad_ram_ref fails then abort_test (bad_ram_ref)
   end if

! Check Data Bus buffers.

   if gfi test "U3-2"  fails then abort_test("U3-2" )
   if gfi test "U23-2" fails then abort_test("U23-2")
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST PROGRAM to test RAM CIRCUIT FUNCTIONAL BLOCKS.          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Setup

   podsetup 'enable ~ready' "on"
   podsetup 'report forcing' "on"
   setspace space (getspace space "memory", size "word")

! Main part of test

   testramfast addr 0, upto $1FFFE, delay 250, seed 1

end program
```

```
program test_rom2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the ROM functional block.                               !
!                                                                            !
! This program tests the ROM functional block of the Demo/Trainer.  The     !
! TL/1 testromfull command is used to test the ROMs.  If the ROMs are       !
! found to be faulty, then one of seven built-in fault conditions is        !
! generated.                                                                 !
!                                                                            !
! TEST PROGRAMS CALLED:                                                      !
!    abort_test (ref-pin)                    If gfi has an accusation,       !
!                                            display the accusation;         !
!                                            otherwise create a gfi hint     !
!                                            for the ref-pin and terminate   !
!                                            the test program (GFI begins    !
!                                            troubleshooting).               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT CONDITION HANDLERS:                                                !
!      Built-in testromfull fault condition handlers                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle rom_sig_incorrect (addr)
   declare numeric addr
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlBAD signature detected,  CONTINUING"
   fault rom_component addr_bits addr
end handle

handle rom_addr_fault (addr)
   declare numeric addr
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRom address line fault detected,  CONTINUING"
   fault rom_component addr_bits addr
end handle

handle rom_addr_addr_tied (addr)
   declare numeric addr
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRom address line tied detected,  CONTINUING"
   fault rom_component addr_bits addr
end handle

handle rom_data_high_tied_all (addr)
   declare numeric addr
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRom data all high detected,  CONTINUING"
   fault rom_component addr_bits addr
end handle

handle rom_data_low_tied_all (addr)
   declare numeric addr
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRom data all low detected,  CONTINUING"
   fault rom_component addr_bits addr
end handle
```

```
handle rom_data_fault (addr)
   declare numeric addr
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRom data line fault detected,  CONTINUING"
   fault rom_component addr_bits addr
end handle

handle rom_data_data_tied (addr)
   declare numeric addr
   declare string clear_screen = "\1B[2J"
   print clear_screen
   print "\nlRom data lines tied detected,  CONTINUING"
   fault rom_component addr_bits addr
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Redirected fault condition handler:                                     !
!                                                                           !
!   Use failing address bits parameter passed from testromfull fault        !
!   condition handlers to gfi test the ROM bank that failed.                !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle rom_component (addr_bits)
   declare numeric addr_bits

   if addr_bits >= $F0000 then
      if gfi test "U27-1" fails then abort_test("U27-11") \ return
      if gfi test "U28-1" fails then abort_test("U28-11") \ return
   else
      if gfi test "U29-1" fails then abort_test("U29-11") \ return
      if gfi test "U30-1" fails then abort_test("U30-11") \ return
   end if
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST PROGRAM to test ROM CIRCUIT FUNCTIONAL BLOCK               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Setup.

   podsetup 'enable ~ready' "on"
   podsetup 'report forcing' "on"
   setspace space (getspace space "memory", size "word")

! Main part of Test.

   testromfull addr $F0000, upto $FFFFE, addrstep 2, sig $156F
   testromfull addr $E0000, upto $EFFFE, addrstep 2, sig $B61E

end program
```

```
program test_rs232b

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the SERIAL I/O functional block.                      !
!                                                                          !
! This program tests the SERIAL I/O functional block of the Demo/          !
! Trainer.  The two RS-232 ports are tested by setting three Dip           !
! Switches to loop back the two ports (SW4-4, SW4-5 and SW6-4 loop back !
! ports A and B).  The SERIAL I/O functional block also outputs two        !
! interrupt request signals.  This program also checks the interrupt       !
! circuitry.                                                               !
!                                                                          !
! TEST PROGRAMS CALLED:                                                    !
!    abort_test  (ref-pin)                  Call fail for reference name  !
!                                           then if gfi has an accusation !
!                                           display the accusation else   !
!                                           create a gfi hint for the     !
!                                           ref-pin and terminate the test!
!                                           program (GFI begins trouble-  !
!                                           shooting).                    !
!                                                                          !
!    frc_int     ()                         POD PROGRAM forces repetitive !
!                                           interrupt acknowledge cycles  !
!                                           and returns first interrupt   !
!                                           vector found on data bus.     !
!                                                                          !
!    rd_cscd     ()                         POD PROGRAM returns the 24 bit!
!                                           interrupt cascade address that!
!                                           was found on the address bus  !
!                                           during the last interrupt     !
!                                           acknowledge cycle.            !
!                                                                          !
!    rd_rearm    ()                         POD PROGRAM returns the most  !
!                                           recent interrupt vector and   !
!                                           rearms the pod to respond to  !
!                                           the next interrupt.           !
!                                                                          !
! FUNCTIONS CALLED:                                                        !
!    sync_buffer (address, data)            Synchronize FIFO buffer in    !
!                                           DUART to be last byte received!
!                                           Receive buffer is located at  !
!                                           the value of address.  The    !
!                                           data in data is written to the!
!                                           DUART and then read until it  !
!                                           appears in the FIFO or count  !
!                                           expires.                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare
   string q                              ! used to get input from keyboard
   string rev  = "\1B[0;7m"              ! Reverse Video escape sequence
   string norm = "\1B[0m"                ! Normal Video escape sequence
end declare
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT HANDLERS:                                                     !
!      These fault conditions are generated by the this program.  These !
!      handlers verify the failure using the Probe or I/O Module and    !
!      then pass control to GFI.                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle 'RS232 Port A failed'
   if gfi test "U11-35" fails then abort_test("U11-35")
end handle

handle 'RS232 Port B failed'
   if gfi test "U11-5" fails then abort_test("U11-5")
   if gfi test "U11-11" fails then abort_test("U11-11")
end handle

handle 'Interrupt failed'
   if gfi test "U10-2" fails then abort_test("U10-2")
   if gfi test "U20-9" fails then abort_test("U20-9")
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONS                                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

function sync_buffer( address, data )
   declare numeric address
   declare numeric data

! Synchronize FIFO buffer in DUART.  Write and then read until correct data
! is returned or count has expired.

   write addr address, data data      ! Transmit Data 31 on port A
   wait time $200
   cnt = 0  \ x = 0
   loop until x = data or cnt > 3
      x = read addr address
      cnt = cnt + 1
   end loop
end function

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the SERIAL I/O Functional Block.                   !
                           !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Set interrupt acknowledge cycles on and use the 80286
! pod specific programs rd_rearm(), frc_int() & rd_cscd().

   podsetup 'report intr' "off"
   podsetup 'intr_ack on'        ! Enable Interrupt Ack. cycles
   option = getspace space "i/o", size "byte"
   setspace (option)
   execute check_loop()
   execute rd_rearm()            ! Clear interrupts
```

```
! Main part of Test.  Verify DUART port A.

   sync_buffer( $2006, $61 )        ! Synchronize FIFO in DUART for port A
   write addr $2006, data $55       ! Transmit Data 31 on port A
   wait time $200
   if ((read addr $2002) and $F) <> $D then fault 'RS232 Port A failed' \ return
   if (read addr $2006) <> $55 then fault 'RS232 Port A failed' \ return
   write addr $2006, data $55       ! Transmit Data 31 on port A
   wait time $200
   if ((read addr $2002) and $F) <> $D then fault 'RS232 Port A failed' \ return
   if (read addr $2006) <> $55 then fault 'RS232 Port A failed' \ return

! Verify DUART port B and interrupts.

   sync_buffer( $2016, $61 )        ! Synchronize FIFO in DUART for port B
   write addr $201E, data $FF       ! set output port low
   write addr $2016, data $31       ! Transmit Data 31 on port B
   if frc_int() <> $22 then fault 'Interrupt failed' \ return
   if rd_cscd() <> $2016 then fault 'Interrupt failed' \ return
   if (readstatus() and 8) <> 8 then fault 'Interrupt failed' \ return
   if (read addr $2016) <> $31 then fault 'RS232 Port B failed' \ return
   if frc_int() <> $27 then fault 'Interrupt failed' \ return
   write addr $201C, data $FF
   if ((read addr $201A) and 2) <> 0 then fault 'RS232 Port B failed' \ return

end program
```

```
program test_video2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the VIDEO functional block                         !
!                                                                      !
! This program tests the VIDEO functional block of the Demo/Trainer.   !
! The video test uses the gfi test command to run stimulus programs and !
! to check the outputs of the Video circuit against the stimulus program!
! response files.  The gfi test command returns a passes status if all  !
! the measured results from running the stimulus programs match the     !
! response files.  Otherwise the gfi test command returns a fails       !
! status.                                                               !
!                                                                      !
! TEST PROGRAMS CALLED:                                                 !
!    abort_test (ref-pin)                   If gfi has an accusation,   !
!                                           display the accusation;     !
!                                           otherwise create a gfi hint !
!                                           for the ref-pin and terminate !
!                                           the test program (GFI begins !
!                                           troubleshooting).           !
!                                                                      !
!    tst_vidctl ()                          Test program to test the video!
!                                           control functional block    !
!                                           outputs.  Returns passes    !
!                                           termination status if       !
!                                           functional block is good else !
!                                           return fails termination    !
!                                           status.                     !
!                                                                      !
!    tst_vidram ()                          Test program to test the video!
!                                           RAM functional block outputs. !
!                                           Returns passes termination  !
!                                           status if functional block is !
!                                           good else return fails      !
!                                           termination status.         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   FAULT CONDITION HANDLERS:                                           !
!      These fault conditions are generated by the this program.  These !
!      handlers isolate the failure in the video circuit to the Video   !
!      control section, Video RAM section or the Video output section.  !
!      Once the failing Video subsection has been identified, then GFI  !
!      is started.                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

handle video_output

   ! IF Video Control section is bad, tst_vidctl will start GFI.

   if tst_vidctl() fails then return

   ! IF Video RAM section is bad, tst_vidram will start GFI.

   if tst_vidram() fails then return

   ! Video Control and Video RAM have passed.  Video Out is bad.  Start GFI.

   abort_test("J3-9")
end handle
```

```
handle video_scan
      gfi hint "J3-8"
      gfi hint "J3-9"
      fault 'gfi hints generated' '  please  run  gfi'
end handle

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the VIDEO Functional Block.                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Setup and initialization.

   connect clear "yes"
   podsetup 'enable ~ready' "on"
   print "\nl\nl"

! Main part of Test.

   if gfi test "J3-8" fails then fault video_scan \ return
   if gfi test "J3-9" fails then fault video_scan \ return

   if gfi test "U78-11" fails then fault video_scan \ return
   if gfi test "U78-28" fails then fault video_output \ return
   if gfi test "U78-29" fails then fault video_output \ return
   if gfi test "J3-7" fails then fault video_output \ return

end program
```

(This page is intentionally blank.)

Section 7
Troubleshooting

After a failing functional block is isolated with a diagnostic program, Unguided Fault Isolation (UFI) or Guided Fault Isolation (GFI) troubleshooting can be used to backtrace to the bad node or component.

UNGUIDED FAULT ISOLATION (UFI) 7.1.

UFI troubleshooting is valuable when you need experience with stimulus programs before expanding to the GFI environment. It lets you use stimulus programs to determine whether a node is good or bad, without having to enter a node list for the UUT.

UFI is used in a manner similar to GFI: the GFI key on the operator's keypad begins the process. Unlike GFI, UFI is designed to test only output pins. When testing with the probe, the output source for a node can be characterized and the other points on the node (such as inputs) can be probed looking for the same response. However, when testing with the I/O module, only the output pins can be measured because the other pins on the node are connected to I/O module pins different from the pins UFI thinks it should be measuring.

When an operator needs to troubleshoot boards before the GFI database is developed, he can use stimulus programs in UFI mode while waiting for GFI to be completed. However, he

needs to understand the UUT since UFI does not recommend the next location to test.

GUIDED FAULT ISOLATION (GFI) 7.2.

The 9100A/9105A's built-in GFI algorithm guides an operator in diagnosing a faulty circuit to the component or node level without assistance from a skilled technician.

Once a functional test or larger diagnostic program has generated a list of suspect nodes, GFI troubleshooting can begin. The GFI key on the operator's keypad starts the process. GFI begins with a bad output and tests the suspect node. Nodes are exercised with a stimulus program and determined to be good or bad by comparing their measured response to responses learned from a known-good UUT.

When a node is bad, GFI tests the inputs which affect that node and recommends which node to test next. If the output of a component is bad and all inputs to the component are good, GFI accuses the component of being bad or the output node of being loaded. The node may be shorted to another node or a defective component may be loading the node. If an input is bad and the output source for that node is good, GFI accuses the node of having an open circuit.

The GFI capability is general enough to troubleshoot most digital circuits. To apply GFI to a particular UUT, however, you will need to supply UUT-specific information to the GFI database for that UUT. The files used for this database are summarized in Section 7.5 of this manual and described fully in the Guided Fault Isolation section of the *Programmer's Manual.*

STIMULUS PROGRAMS 7.3.

Stimulus programs are TL/1 programs used by GFI or UFI to exercise UUT nodes in such a way that responses at the nodes can be analyzed and compared to responses of nodes on a

known-good UUT. A typical stimulus program consists of up to 6 main parts:

1. *(As required)* - Initialize the UUT and define the measurement device.

2. *(As required)* - Setup of the pod, probe, or I/O module.

3. Use the *arm* command to start the measurement of the node response.

4. Use any commands necessary to apply the stimulus.

5. Use the *readout* command to end the measurement of the node response.

6. *(As required)* - Restore any conditions altered by the setup step above (step 2).

Stimulus programs should satisfy three very important criteria:

- The program must be independent, initializing the UUT as required. This is because GFI can begin backtracing at any node, and the state of the UUT, prior to running the stimulus, is unknown. The program must also restore any adjustments it makes to the calibration offset.
- During stimulus execution, only one pin should drive a node: that is, during the period between the arm and readout commands, one and only one pin should be a node signal source (data should flow in only one direction).
- There should be at least one stimulus program for each output to the node.

See the "Stimulus Programs" section in the *Programmer's Manual* for more detailed information on stimulus programs.

STIMULUS PROGRAM RESPONSES 7.4.

Both UFI and GFI select the appropriate stimulus programs to exercise a node to be measured and compare the actual response at the node with a stored response from a known-good UUT. These responses may be any of the following (or combinations of them):

- CRC Signature.
- Transition Count.
- Frequency.
- Asynchronous Level History.
- Synchronous Level History.

The information below summarizes each of these response measurements. See the Guided Fault Isolation section of the *Programmer's Manual* for more complete information.

Learning Responses From a Known-Good UUT 7.4.1.

The 9100A editor's LEARN function is used to learn a set of responses measured on known-good UUT nodes. Once a stimulus program is written to exercise a node, a response file can be generated. To do this, the 9100A is commanded to learn responses at a node or set of nodes and the system prompts the operator to connect the measurement device (probe or I/O module) to the component providing the node signal source. The 9100A makes a series of measurements and determines the characteristics. It learns the response with three measurements (early, normal, and late clock or sync events) to make sure the response is stable and that the measurement can be used as a reliable characterization of that node.

Node characterization may use one or more of five characteristics to determine whether the node is good or bad. You can select which of the five should be saved in a response

file. GFI and UFI use these saved characteristics to determine whether a node is good or bad.

CRC Signatures 7.4.2.

It is very important to ensure that a CRC signature used in node characterization will properly identify all good UUTs, at all measurement temperatures and power supply levels. A marginal signature occurs when the measured node changes state near the clock transition or when the Start, Stop, or Clock signals are not stable. A marginal signature may appear stable on one UUT and thereby lead to a false sense of security. Other UUTs may yield different signatures because of temperature or power supply variations.

When the 9100A editor learns a signature, it attempts to identify marginal CRCs by collecting signatures with advanced clock edges, normal clock edges, and delayed clock edges. If a signature has the same value for advanced and normal clock edges, it will be suffixed by a "-" sign. If a signature has the same value for normal and delayed clock edges, it will be suffixed by a "+" sign. If all three values agree, the signature is displayed with no qualification.

A variable signature results if the Start, Stop, or Enable signals are irregular, compared to the Clock signal. In addition, since the Start, Stop, and Clock signals are edge-triggered, unstable signatures will result if the Start or Stop signal edge occurs at the same time as the Clock signal edge.

Figure 7-1 shows how to test whether the start/stop interval is stable. Connect the Clock to the clock signal you want to use. Connect the probe or I/O module to a logic-high level and connect the Start and Stop lines to the locations where you would connect these lines when making the signature measurement. If the start/stop interval is stable, a constant number of clocks will occur between the start and stop condition, and the signature will be constant. If the CRC signature is not constant, the start/stop interval is unstable.

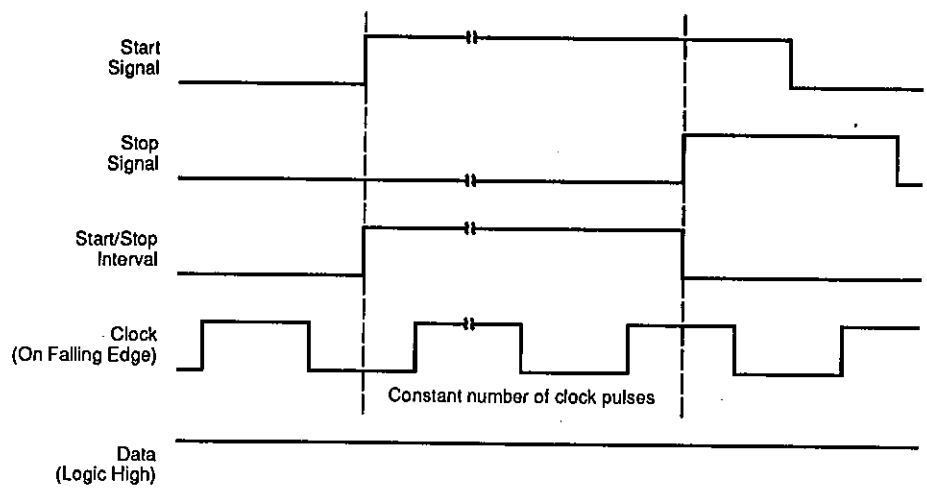


Figure 7-1: Testing for Start and Stop Stability

Unstable signatures may also be caused by Start or Stop signal edges which occur at the same time as the Clock signal edge or by Start or Stop signals which are asynchronous to the selected clock signal. Use an oscilloscope to determine whether a line is irregular or whether a timing problem exists between the Clock signal and the Start or Stop signal.

If unstable signatures are caused by Start or Stop signal edges which occur at the same time as the Clock signal edge, select the other Clock edge (+ or -) and use the *getoffset* and *setoffset* TL/1 commands to adjust the measurement timing.

Other Characterizations 7.4.3.

Some circuits are difficult to characterize by a CRC signature. The node may have regular activity but there might be no signal which can be used as a clock to gather a consistent signature. In many such cases, nodes can be characterized by using transition counts.

The transition count works on asynchronous signals. The transition count can monitor information that the CRC will not detect, such as extra transitions between CRC clocks. The transition count will typically be a range of counts, defined by a minimum and maximum, that represents the extremes of the three measurements taken by the LEARN function. Only low-to-high transitions are counted (not high-to-low). When the measurement is synchronized to the external lines, the data input is gated with the enable line, if used. A count of zero will result if the enable-true window does not overlap the low-to-high transition of the data.

The frequency of a signal may be more important than its CRC or transition count. This is especially true for system clocks. If a system clock is run at 4 MHz rather than 8 MHz, everything on the board could appear to be good. However, when the board is plugged into a system, the board running at 4 MHz may cause a system failure. Frequency is also important for video signals such as horizontal and vertical sync.

Level history is an important characterization parameter when combined with signatures or transition counts. If a faulty node has the correct timing but swings between ground and an invalid level for part of the time, measuring asynchronous level history would detect this fault, which will be missed if only a CRC is measured.

Consider the case where a node that should go high and low is stuck on a faulty UUT. Using both CRC and asynchronous level history to characterize the node will provide more complete information to the technician who repairs the board. The operator can see that the line is stuck when it should be changing.

Level history can be used to detect glitches. If the measurement period is set so that a signal is either high or low during measurement, with no glitches, the level history will show only high or low. If the level history shows both high and low, a glitch has occurred.

Calibration of the I/O Module and Probe 7.4.4.

Whenever the pod performs a microprocessor operation, it generates a synchronization pulse which the 9100A/9105A uses to measure signatures and clocked levels. The synchronization pulse can be generated by several devices, including the pod or an external clock.

In order for the system to measure critical signals reliably, each measurement device (I/O modules and probe) must be calibrated to this synchronization pulse on the system where it will be used, since each measurement device contains its own electronics that affect timing. If your tests must be accurate to within a few tens of nanoseconds on signal edges, calibration should be done.

The procedures for calibration are given in the *Technical User's Manual*. Calibration should be performed for each measurement device and for each synchronization mode of that device on the particular 9100A/9105A system where it will be used. For

example, the probe for an 80286-based UUT should be calibrated to EXT, POD ADDR and POD DATA on the 9100A/9105A where the probe will be used.

Calibration is UUT-dependent. For this reason, calibration settings should be saved under the specific directory for that UUT. If calibration is not performed, default calibration values will be used. These default calibration values will only work properly in some UUTs (those which have ample timing margin or which operate at slow speeds).

Adjusting Sync Timing 7.4.5.

The sync pulse that the measurement devices (I/O modules and probe) receive from the 9100A/9105A comes either from the pod or an external clock signal. The pod may provide sync pulses with different timings relative to microprocessor read/write operations, depending on the synchronization mode of the pod. For example, the 80286 pod has POD ADDR and POD DATA sync modes. The sync pulse in POD ADDR mode is earlier than in the POD DATA mode. See the timing diagram in the pod manual for the pod you are using.

Most signals on a UUT can be characterized using the external or pod sync mode. However, in some cases, the sync pulse occurs at a different time than when the signal should be measured.

The *getoffset* and *setoffset* TL/1 commands can be used to adjust the time when a signature or clocked level measurement is made, relative to the sync pulse. Figure 7-2 shows how this offset is implemented in the probe or the I/O module. The data to be measured passes through one delay line and the sync pulse passes through a different delay line. One of the delay lines is variable. By adjusting the variable delay line, the data is measured at a different time relative to the sync pulse.

Section 3 of the *TL/1 Reference Manual* contains details about the *getoffset* and *setoffset* commands, including the approximate timing resolutions of the probe and the I/O module.

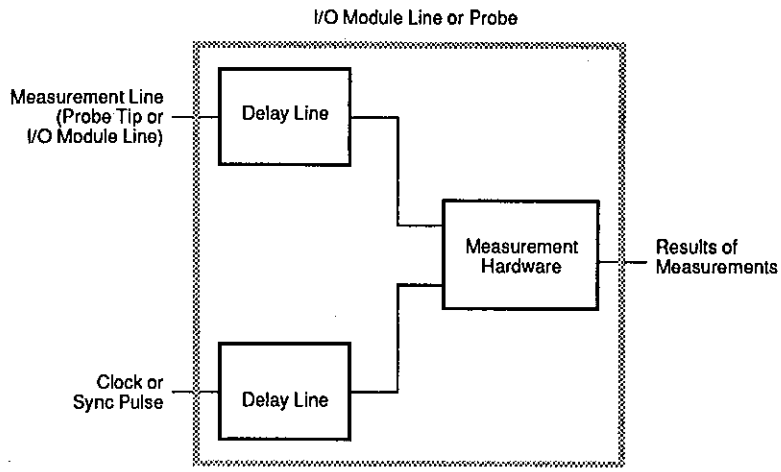


Figure 7-2: Synchronization-Pulse Delay Mechanism

Appendices C and E of the *Technical User's Manual* contain additional timing specifications for the pod, probe, and I/O modules. The *Supplemental Pod Information for 9100A/9105A Users* manual and the pod manuals have more detailed information about pods.

When a program adjusts the sync timing, the original timing should be restored at the end of the program. This can be done by storing the result of a *getoffset* command, adjusting the timing with *setoffset*, and readjusting the timing with *setoffset* at the end of the program with the stored *getoffset* value.

Dynamic RAM circuits usually require sync timing adjustment in order to measure the RAS and CAS signals, which do not necessarily coincide with the POD ADDR or POD DATA sync pulses. The Demo/Trainer UUT stimulus programs for the Dynamic RAM Timing functional block show one way to adjust the sync timing.

THE UUT DESCRIPTION 7.5.

The UUT description, which provides the 9100A/9105A with information used for GFI and UFI, consists of:

- Reference designator list (reflist).
- Part Library (part descriptions). A basic part library is provided with the system.
- Node list (net list or wire list).

The *Programmer's Manual* provides detailed information about this database and how GFI and UFI use it. The following sections are simply a brief overview.

Reference Designator List (REFLIST) 7.5.1.

The reference designator list establishes the relationship between reference designators (such as "U80") and a part or component

type (such as 7410). It also specifies the testing device (probe or I/O module) to be used on the component.

A sample Demo/Trainer UUT reference designator list is shown in Appendix A. GFI and UFI both require the reference designator list to determine the device needed to test a component.

No distinction is made between families of components, such as 74LS00 or 74HCT00. The Fluke-supplied part library uses generic names like 7400 and 7432, so when you make entries in a reference designator list you will need to use generic names.

Part Library (Part Descriptions) 7.5.2.

The part library is a group of files (part descriptions) that describe UUT components. A part description specifies each pin to be an input, output, bidirectional, ground, power, or unused. Each output has a list of related inputs which affect that output. The library can be accessed through any UUT directory. A basic part library is supplied by Fluke. You can add part descriptions, including custom designs.

See the Guided Fault Isolation section of the *Programmer's Manual* for examples of part descriptions.

Node List (Net List or Wire List) 7.5.3.

The node list specifies interconnections between reference designators. The list is only necessary for GFI, which uses it to backtrace between components.

A complete node list contains one line for each node in a UUT. The pins on one line are all connected to form a node. Lines may be continued on the next line with the backslash (\) character.

Appendix B contains a node list for the Demo/Trainer UUT. Reviewing this example will be helpful to you when developing you own node lists.

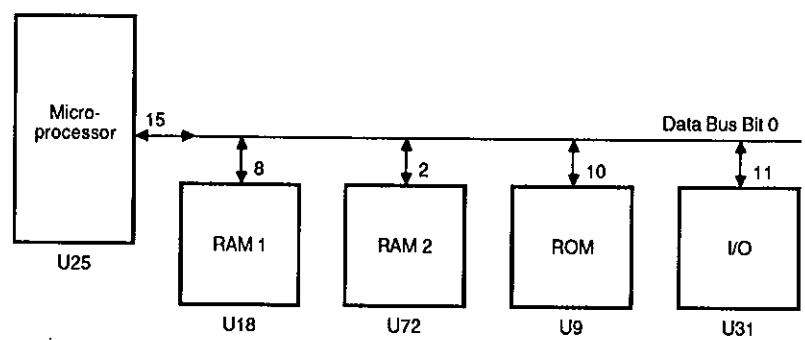
Bus-Master Pins in a Node List 7.5.4.

The 9100A normally determines the flow of data from the node list; it assumes that data can be sent from any pin to any other pin on a given node. However, sometimes two pins are connected together by a node but do not actually communicate with each other; this situation commonly arises in bus-oriented systems with many components connected to a common microprocessor data bus.

In such cases, you need to let GFI know that only some pins (called bus-master pins) can communicate with all the other pins on the same node. This is done by entries in the optional **masters* section of the node list.

The **masters* section is optional, and for most UUT node lists it can be omitted. Where it is needed, it usually contains just a short list of pins, because most nodes have only a single source. It is only for nets such as the one in the following example that the **masters* section becomes important.

Consider the node shown below: It consists of bit 0 of a bidirectional data bus connecting several components to a microprocessor.



Only pin U25-15 can talk to all other input pins on the node and only U25-15 can receive from all other output pins on the node. Either condition would be sufficient to make U25-15 a bus-master pin.

For this reason, pin U25-15 is shown as a bus-master pin in the partial node list below. It is listed in the regular section of the node list and is also included in the optional **masters* section of the node list.

```
.
.
.
U8-12 U3-9 U42-21
U25-15 U19-8 U22-2 U9-10 U31-11
U17-4 U28-5 U27-6
.
.
.
*masters
U25-15
```

See the Node List section in the *Programmer's Manual* for more information about bus-master pins.

Choice of Backtracing Path 7.5.5.

If there are two or more stimulus programs available for a node, GFI will attempt to use the program that stimulates all of the node's outputs (and related inputs) before using programs that stimulate only some of the node's pins.

Here are three cases that relate to the AND gate in Figure 7-3. Each case shows the test results from two stimulus programs, A and B, and the conclusion that GFI comes to:

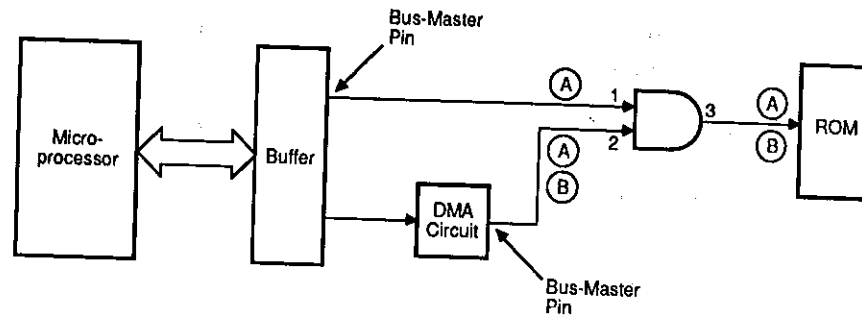


Figure 7-3: Direction-Control Example

Case 1:	*Input 1*	*Input 2*	*Output 3*
Stimulus Program A	good	good	bad
Stimulus Program B	–	bad	bad

GFI will accuse the node of being bad because stimulus program A covers all the nodes and is therefore evaluated first. In this case stimulus program B will not be executed.

Case 2:	*Input 1*	*Input 2*	*Output 3*
Stimulus Program A	bad	good	bad
Stimulus Program B	–	bad	bad

GFI will test the component connected to input 1, again because stimulus program A covers all the nodes and is therefore evaluated first Therefore, GFI will backtrace to the Bus Buffer.

Case 3:	*Input 1*	*Input 2*	*Output 3*
Stimulus Program A	good	good	good
Stimulus Program B	–	bad	bad

GFI will test the component connected to input 2, because stimulus program A finds no problem and the system goes on to evaluate stimulus program B. Therefore, GFI will backtrace to the DMA circuit.

Consider these two problems in Figure 7-3, in which both the microprocessor and the DMA controller are both **master* components:

- If the problem is in the microprocessor, evaluation is the same as for Case 2, above, and GFI troubleshooting traces back to the microprocessor from input 1 of the AND gate.
- If the problem is in the DMA controller, evaluation is the same as for Case 3, above, and GFI troubleshooting traces back to the DMA circuit from input 2 of the AND gate.

While you can effectively steer GFI by designing stimulus programs to cover *all* or only *some* inputs and outputs, you do not usually need to worry about control of the backtracing path; it is only needed in special circumstances.

Normally, you should design stimulus programs that test *all* inputs and outputs of a node or component. If there is no single stimulus program that covers all inputs and outputs, the 9100A/9105A uses these criteria to determine status:

- If ANY stimulus program gives a BAD response on a pin, the pin is considered BAD.
- If ALL stimulus programs give GOOD responses on the pin, the pin is considered GOOD.
- Otherwise, the pin is considered UNKNOWN.

SUMMARY OF GFI COVERAGE 7.6.

The 9100A provides a convenient means to check the completeness of the information you have entered into the GFI database for a particular UUT. When viewing the UUT directory display, you can press the SUMMARY softkey to request generation of a summary of GFI coverage for that particular UUT. The compiled database (GFIDATA or UFIDATA) will be examined and a summary will be generated, displayed on the monitor, and stored in a UUT text file that you specify. If you press the Shift key on the programmer's keyboard and the SUMMARY softkey, the summary will appear on the monitor without sending a copy to a text file.

Creating a Summary of GFI Coverage

The following procedure is used to generate a Summary of GFI Coverage for a UUT:

1. Press the EDIT key on the operator's keypad to enter the Editor (unless you are already in the Editor).

2. Use the EDIT key on the Programmer's Keyboard to enter the name of the UUT so that the UUT directory for this UUT is displayed on the monitor. The UUT directory you have selected must contain a compiled database (either GFIDATA or UFIDATA).

3. Press the SUMMARY Softkey (F8) and the 9100A will issue the prompt shown below to ask for a text file name:

```
Generate GFI Summary to TEXT file _____
```

The Summary of GFI Coverage to be generated will be stored in this text file.

4. Type in the text file name you wish and press the Return key. The 9100A will then begin generating the Summary of GFI Coverage for the UUT and will display the results on the monitor.

When the generation is complete, the following message will appear on the monitor:

```
Press Msgs key to continue
```

When you press the Msgs key on the programmer's keyboard, the UUT directory display will reappear on the monitor. You can use the Edit key on the programmer's keyboard to access the text file you generated.

Statistical Summary

The first part of the Summary of GFI Coverage is a statistical summary of the UUT, based on the GFI database you have provided. Figure 7-4 shows a typical example of such a summary. Each entry in the summary is described below:

- **Summary for /<disk drive>/<UUT>:** In Figure 7-4, HDR is the disk drive and the UUT directory name is EXAMPLE.
- **Parts:** The number of unique part types in the UUT, based on the reference designator list.
- **Reference Designators:** The number of reference designators in the UUT, based on the node list.
- **Connected Pins:** The number of UUT pins that are connected to other pins on the UUT, based on the node list.
- **Unconnected Pins:** The number of UUT pins that are not connected to any other UUT pins, based on the node list.
- **Total Pins:** The total number of pins on the UUT.
- **Programs:** The number of TL/1 programs that can be used by GFI as stimulus programs. This number is equal to the number of response files.
- **Testable Connected Pins:** The number of connected pins that can be tested by GFI. Testable pins have either been characterized with LEARN, or are a member of a node that has been characterized with LEARN.
- **Testable Unconnected Pins:** The number of unconnected pins that can be tested by GFI. Testable unconnected pins have been characterized by LEARN and appear in a response file.
- **Total Testable Pins:** The total number of UUT pins that can be tested with GFI, given the database you have entered.

```
Summary for /HDR/EXAMPLE:

     53  Parts
    167  Reference Designators
   1694  Connected Pins
    225  Unconnected Pins
   1919  Total Pins
     42  Programs

   1688  Testable Connected Pins
     16  Testable Unconnected Pins
   1704  Total Testable Pins

      6  Untestable Connected Pins
    209  Untestable Unconnected Pins
    215  Total Untestable Pins

    99% Test Coverage of Connected Pins
    88% Test Coverage of Total Pins
```

Figure 7-4: Statistical Summary Display for a UUT

- **Untestable Connected Pins:** The number of connected pins that cannot be tested with GFI, due to an incomplete database.
- **Untestable Unconnected Pins:** The number of unconnected pins that cannot be tested with GFI, due to an incomplete database.
- **Total Untestable Pins:** The total number of UUT pins that cannot be tested with GFI, given the database you have entered.
- **Test Coverage of Connected Pins:** The percentage of connected pins on the UUT that can be tested with GFI, given the database you have entered. A figure of less than 100% indicates an incomplete database.
- **Test Coverage of Total Pins:** The percentage of UUT pins that can be tested with GFI, given the database you have entered. This figure is typically less than 100% because a UUT often has unused pins.

Pin Coverage

The second part of the GFI Summary of Coverage display is a matrix showing how component pins are tested with the database you have provided. Figure 7-5 shows a partial example of a pin coverage matrix. The matrix is organized with the reference designators listed vertically (in the left-most column) and with component pin numbers listed horizontally. The number of pins per line will be the number required by the largest component in the list. If more than 35 pins are required, the display will produce a second list of reference designators following the first list and this second set will have pin numbers starting with 36 and continuing up from there.

Each component pin has a one-character symbol that shows how GFI looks at the pin given the database you have provided. The table at the bottom of Figure 7-5 shows the meaning of each symbol that is possible:

```
Pin Coverage:

                       1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
     1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5

C15  I O . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
C16  I I . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
C17  I O . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 J5  I * * I I * * * I * * I I I I I I I I I I I I I I I I I I I I I I I I
 J6  I I I I I . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 Q1  O I I . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 Q2  O I I . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
R10  O I . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
R11  I O . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
R12  I O . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 S1  I I . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 S2  I I . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
U10  I I I I I B I I B I I B I I B B I I B I . . . . . . . . . . . . . . .
U11  * I * I I I I * I I I * O O O * O B B B B I * O B B B B * * * * O * I
U12  O O I O I O O I O I I O I O I I . . . . . . . . . . . . . . . . . . .
U13  I O I O I O G O I O I O I P . . . . . . . . . . . . . . . . . . . . .
U14  O * * O O * * * I * * O O O O O O O O O O O O O O O O O I I I O O O I
```

Symbol	*Meaning*
I	The pin is testable as an input only.
O	The pin is testable as an output only.
B	The pin is testable as both an input and an output.
P	The pin is testable as a power pin.
G	The pin is testable as a ground pin.
*	The pin is not testable (because it has no associated stimulus program or no known-good response stored for this pin).
	There is no such pin in the database.

Figure 7-5: Pin Coverage Display for a UUT

FAULT CONDITION EXERCISERS 7.7.

When the 9100A/9105A detects a fault, and a fault condition handler is not defined for the fault condition raised, a fault message will appear on the operator's display. At this point, the operator can press the LOOP key on the operator's keypad to repeatedly reproduce the fault so that it can be isolated manually. To do so requires that a fault condition exerciser exist for the fault condition that was raised. If the exerciser exists, it is invoked continually until the operator presses the STOP key on the operator's keypad.

A fault condition exerciser is a software block designed specifically to reproduce a fault condition in a UUT. Two types of exercisers are available: built-in exercisers and user-defined exercisers.

When a fault condition is raised by a built-in stimulus function (such as read, write, ramp, toggle, or rotate) or a built-in test function (such as *testbus, testramfast, testramfull,* or *testromfull*), the 9100A/9105A has a pre-defined sequence of commands that exercise the fault when the LOOP key is pressed. These are called built-in fault condition exercisers. In addition, you as a programmer can write your own fault condition exercisers for fault conditions that you define or to replace the built-in fault condition exercisers. When one of these fault conditions is encountered and the LOOP key is pressed, the fault condition exerciser with the matching name is invoked.

If a fault condition exerciser for the displayed fault condition is found when the LOOP key is pressed, the fault condition exerciser is invoked repeatedly to stimulate the UUT. This allows the probe to be used to examine node responses in the circuit and to trace faulty circuit operation to its cause.

REPAIR AFTER TROUBLESHOOTING 7.8.

When GFI terminates, it will often display one of the following messages:

- Open circuit.
- Bad IC or output loaded.

When GFI reports an *open circuit*, it has found an input which is bad even though the signal source on that node is good. To repair the node:

1. Retest both ends of the node to make sure the output was properly probed.
2. Confirm the open circuit with an ohmmeter.
3. Trace along the node with the ohmmeter until the open point is found.
4. If the node is connected properly, check for:
   - An error in the node list entry for the failed node.
   - Marginal measurements due to the frequency or timing of signals on the node. Ringing may be occurring on the node, or the time between the sync and the signal transitions may be marginal. Change the stimulus setup or the sync timing to correct the problem (see Section 8.5 on adjusting sync timing).

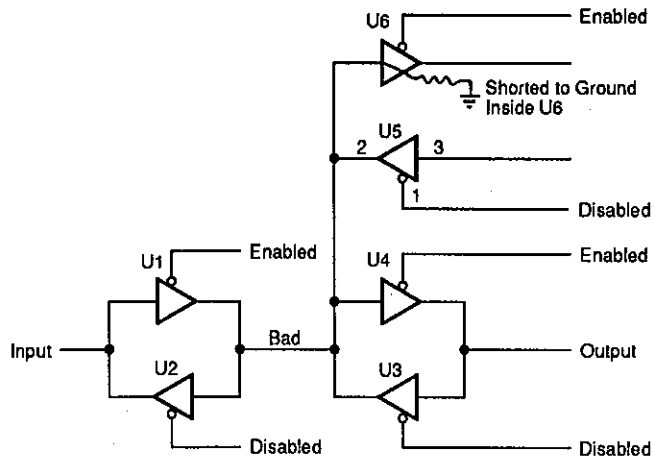
When GFI reports a *bad IC* or *output loaded*, it has found all good inputs and one or more bad outputs. In this case, determine whether the part is bad or the output is loaded. To do this, test the component by overdriving its inputs with the I/O module while measuring level history or CRC signatures.

In doing so, determine whether:

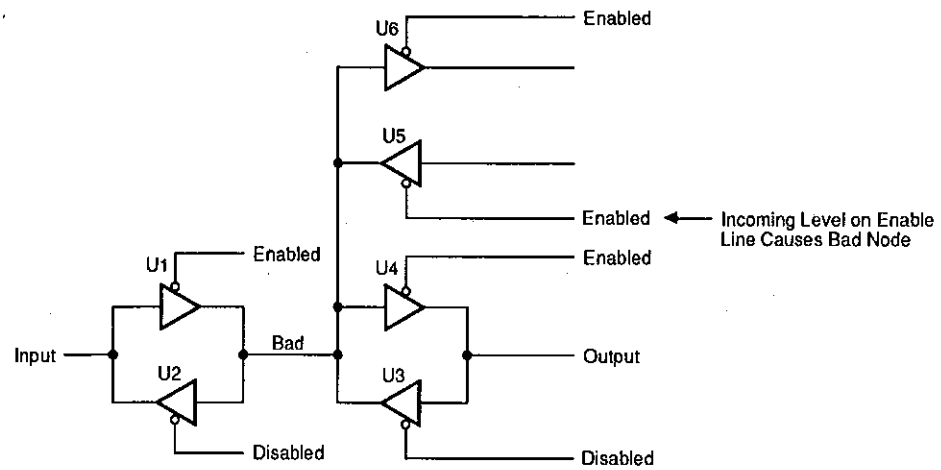
- The level history showed that the line went to a high and low state. If so, the node is only loaded part of the time, or the component is bad.
- The node is loaded. If the component is good but the node is bad, the node must be loaded. The cause of a loaded node can be:

    - A short to another node, the power supply, or ground.
    - A damaged IC loading the node. Example 1 in Figure 7-6 shows a bad input at U6 causing node A to be loaded.
    - Another output source is also driving that node. Check the enable and control lines of any other devices that can drive the node. Example 2 in Figure 7-6 shows node A to be loaded because both U1 and U5 are attempting to drive the node at the same time. U1 is operating as it should but the U5 enable-line state is incorrect and U5 is also driving the same node.

Operators should be provided with a procedure for tracing short circuits. For example, a milliohmmeter can be used to determine the point at which a node is shorted. To do this, attach one lead of the milliohmmeter to the faulty node. With the other lead, look for low resistance paths.



Example 1: Bad IC



Example 2: Bus Contention

Figure 7-6: Node Loading

Section 8
Glossary

If you cannot find a term in the glossary, search the index for a reference to that term.

Active Edge
A signal transition used to initiate action.

Address Space
A section of memory reserved for a particular use, such as the stack. The term "decoded address space" implies memory residing in physically separate chips (selectively enabled by a "decoder"), such as a frame buffer, character generator, or the control registers inside a peripheral chip.

Aliasing
A condition where a component address responds to more than one combination of address bus bits.

Assert
To cause a signal to change to its logical "true" state.

Asynchronous
Not synchronized to the microprocessor or not synchronous to any clock signal.

Automated Test
An automated activity that verifies the correct operation of a circuit by comparing its output to the expected output.

Automated Troubleshooting
An automated process of locating a fault on a UUT.

Backtracing
A procedure for locating the source of a fault on a UUT by checking logic along a logical path from bad outputs to bad inputs until the point where no bad inputs are found.

Bus
A group of functionally similar signals.

Bus Contention
A situation where two or more bus devices are trying to put different data onto the same bus.

CAD
An acronym for Computer-Aided Design. CAD systems let the user create, manipulate, and store designs on a computer.

Comment
Text in a program that is not executed. A comment in a TL/1 program or a node list must begin with an exclamation point (!).

Component
A passive or active part on a UUT.

Control Line
A signal that comes out of a microprocessor and is used to control the UUT.

CRC Signature
CRC is an acronym for Cyclic Redundancy Check. A CRC signature is a compression of a long data stream into a 16-bit number.

Cursor
A symbol on a display (usually a box or an underscore) that indicates where a typed character will appear.

Data Bus
A set of signal paths on which parallel data is transferred between two or more devices.

Device
1. Refers to the probe, an I/O module, a reference designator, or the pod. 2. Also used with I/O operations to specify a port or a disk drive.

DIP
An acronym for Dual In-line Package. A DIP has an equal number of pins on each of its long sides. See also SIP.

Directory
A collection of related sets of data (files, for example) on a disk.

Drivability
Testing whether lines can be driven to the appropriate active high or active low level.

Dynamic Coupling
Data in one memory location is affected by combinations of data in other memory locations.

Edge
The transition from one voltage level to a different voltage level.

Exerciser
See Fault Condition Exerciser.

External Synchronization
Synchronizing a node response measurement using signals external to the pod.

Fault
A defect in a UUT that causes circuitry to operate in a manner that is inconsistent with its design.

Fault Condition
A recognition by the 9100A/9105A that a fault exists on the UUT.

Fault Condition Exerciser
A group of statements that attempts to repetitively reproduce the conditions that generate a fault condition. (Sometimes called just an "exerciser.")

Fault Condition Handler
A group of statements that is executed when a particular fault condition occurs. (Sometimes called just a "handler.")

Fault Condition Raising
The generation of a fault condition either from detecting a fault on a UUT or from using a TL/1 *fault* statement.

Feedback Loop
A circuit in which one or more outputs is routed to the circuit's input.

Forcing Line
Input to the microprocessor that forces it to a particular known state.

Functional Test
An activity that verifies the correct operation of a circuit by comparing its output to the expected output.

GFI
See Guided Fault Isolation.

GFI Summary
A record of the components that have been tested by GFI.

Go/No-Go Test
A pass/fail test; either a unit passes or it doesn't.

Guided Fault Isolation
An algorithm that uses backtracing to troubleshoot a UUT.

Handler
See Fault Condition Handler.

Hexadecimal
Pertaining to the base 16 numbering system. (Often abbreviated as "hex.")

I/O
An abbreviation for Input/Output. The transfer of data to and from devices other than the local memory of the microprocessor system.

I/O Module
An option for the 9100A/9105A that allows simultaneous stimulus or response for multiple points on a UUT.

Level History
A character string that represents a record of the logic levels measured at a point over a period of time. "1", "X", and "0" represent high, invalid, and low states, respectively.

Library
A directory that contains a collection of only a particular type of file. The 9100A/9105A uses four libraries: a part library, a program library, a pod library, and a help library.

Mask
A value where each logic "1" represents a bit that is to be acted on.

Monitor
A 24-line, 80-column video display that connects to the rear panel of the 9100A/9105A.

Node
A set of points that are all electrically interconnected.

Node List
A file containing a description of the interconnection of all pins on a UUT.

Operator
1. A symbol that acts on one or more values or expressions to produce another value. 2. A person who uses the 9100A/9105A for testing or troubleshooting.

Operator's Display
Three-line display on the mainframe of the 9100A/9105A.

Operator's Interface
The operator's display and the operator's keypad.

Operator's Keypad
The set of keys on the front panel of the base unit of the 9100A/9105A.

Overdrive
To put a logic state on a signal line by applying more power than the normal driver for that node. This is how the 9100A/9105A injects signals into the UUT.

Part Description
A file that describes a component on a UUT.

Part Library
A library of part descriptions.

Pod Library
A library of pod descriptions, each of which contains a pod database and pod-related TL/1 programs.

Pod Synchronization
Synchronizing a node response measurement using signals generated by the pod to indicate the sampling time.

Priority Pin
A pin that the GFI program will test first if a particular node is bad.

Probe
A hand-held device that can stimulate and measure any single point on the UUT.

Program Library
A library of programs that can be called by any program in the userdisk.

Programmer's Interface
The monitor and the programmer's keyboard.

Programmer's Keyboard
The keyboard that connects to the side panel of the 9100A.

Raise
See Fault Condition Raising.

Reference Designator
A one to ten character string naming a component on the UUT.

Related Input Pin
An input pin on a component that affects an output pin on that same component.

Response File
A file containing data generated by executing a specific stimulus program to a UUT and recording the responses from its execution.

RUN UUT Test
A feature that allows the normal operation of a UUT using its own program.

Signature
See CRC Signature.

SIP
An acronym for Single In-line Package. See also DIP.

Softkey
A key that has its function determined by software.

State Machine
A circuit which produces output signals in response to input signals and its own internal state. Typically used to generate a sequence of control signals, as in a bus interface.

Stimulus Program
A program that exercises a circuit while the activity on circuit nodes are recorded to see if the circuit produces the same response as a known-good circuit.

String
A group of characters enclosed in double-quote characters (") and manipulated as a single entity.

Synchronous
Coordinated to the transitions of a clock signal.

Termination Status
An indication of whether a UUT passed a test.

Timeout
A condition in which an expected event has not occurred within the expected time period.

Toggle
Change to the complementary logic state.

Transition Count
A record of the number of times the logic level at a node changes from low to high within a period of time.

Troubleshooting
A process of locating the area of a UUT that is causing a fault.

Userdisk
1. A diskette containing test programs and information about a particular UUT. 2. The current disk drive that is used as a source for UUT programs and data.

UUT
Unit Under Test. A physical item, i.e., a board or a system to be tested.

UUT Directory
A set of files that contain information about a particular UUT.

Wait State
A bus cycle which is too short for a slow chip is lengthened by the insertion of one or more clock cycles, called wait states.

Watchdog Timer
A circuit which produces a signal, typically a reset or high-priority interrupt, if a timeout condition is met. For example, an excessive number of wait states may trigger a watchdog timer.

Wildcard
A symbol that represents any sequence of characters. The 9100A/9105A uses the asterisk character (*) for this purpose.

Window
An area of the monitor reserved for certain information to be displayed.

(This page is intentionally blank.)

Appendix A
Demo/Trainer UUT Reflist

NAME: REFLIST
DESCRIPTION:

SIZE: 3,555 BYTES

REF	PART	TESTING DEVICE
R72	RESISTOR	PROBE
R73	RESISTOR	PROBE
R4	RESISTOR	PROBE
R79	RESISTOR	PROBE
R78	RESISTOR	PROBE
R61	RESISTOR	PROBE
R62	RESISTOR	PROBE
R63	RESISTOR	PROBE
R64	RESISTOR	PROBE
R65	RESISTOR	PROBE
R70	RESISTOR	PROBE
C4	CAPACITOR	PROBE
C5	CAPACITOR	PROBE
C8	CAPACITOR	PROBE
C9	CAPACITOR	PROBE
C13	CAPACITOR	PROBE
C15	CAPACITOR	PROBE
C16	CAPACITOR	PROBE
C17	CAPACITOR	PROBE
U74	2016	I/O MODULE
U85	2016	I/O MODULE
U72	2674	I/O MODULE
U78	2675	PROBE
U11	2681	PROBE
U77	27128	I/O MODULE
U30	27256	I/O MODULE

U29	27256	I/O MODULE
U28	27256	I/O MODULE
U27	27256	I/O MODULE
Q1	TRANSISTOR	PROBE
Q2	TRANSISTOR	PROBE
C1	CAPACITOR	PROBE
R35	RESISTOR	PROBE
R1	RESISTOR	PROBE
R77	RESISTOR	PROBE
R80	RESISTOR	PROBE
R15	RESISTOR	PROBE
R14	RESISTOR	PROBE
R16	RESISTOR	PROBE
R13	RESISTOR	PROBE
R17	RESISTOR	PROBE
R12	RESISTOR	PROBE
R18	RESISTOR	PROBE
R11	RESISTOR	PROBE
R27	RESISTOR	PROBE
R25	RESISTOR	PROBE
R24	RESISTOR	PROBE
R28	RESISTOR	PROBE
R29	RESISTOR	PROBE
R23	RESISTOR	PROBE
R30	RESISTOR	PROBE
R19	RESISTOR	PROBE
R68	RESISTOR	PROBE
R69	RESISTOR	PROBE
R20	RESISTOR	PROBE
R21	RESISTOR	PROBE
R22	RESISTOR	PROBE
R34	RESISTOR	PROBE
R33	RESISTOR	PROBE
R3	RESISTOR	PROBE
R5	RESISTOR	PROBE
R6	RESISTOR	PROBE
R7	RESISTOR	PROBE
R8	RESISTOR	PROBE
R32	RESISTOR	PROBE
R31	RESISTOR	PROBE
R26	RESISTOR	PROBE
R9	RESISTOR	PROBE
R2	RESISTOR	PROBE
U34	4164	I/O MODULE
U35	4164	I/O MODULE
U36	4164	I/O MODULE
U37	4164	I/O MODULE
U38	4164	I/O MODULE
U39	4164	I/O MODULE
U40	4164	I/O MODULE

U41	4164	I/O MODULE
U48	4164	I/O MODULE
U49	4164	I/O MODULE
U50	4164	I/O MODULE
U51	4164	I/O MODULE
U52	4164	I/O MODULE
U53	4164	I/O MODULE
U54	4164	I/O MODULE
U55	4164	I/O MODULE
R67	RESISTOR	PROBE
C6	CAPACITOR	PROBE
C7	CAPACITOR	PROBE
R71	RESISTOR	PROBE
R10	RESISTOR	PROBE
R66	RESISTOR	PROBE
U14	80286	PROBE
J5	CONN68	PROBE
U1	82284	I/O MODULE
U15	82288	I/O MODULE
U31	8255	I/O MODULE
U58	7400	I/O MODULE
U24	7400	I/O MODULE
U5	7400	I/O MODULE
U64	7402	I/O MODULE
U57	7404	I/O MODULE
U19	7404	I/O MODULE
U4	7408	I/O MODULE
U63	7408	I/O MODULE
U56	7410	I/O MODULE
U21	74138	I/O MODULE
U8	74138	I/O MODULE
U9	74138	I/O MODULE
U3	74245	I/O MODULE
U23	74245	I/O MODULE
U44	7474	I/O MODULE
CR1	DIODE	PROBE
J2	CONN_RS232	PROBE
J3	CONN_VIDEO	PROBE
J6	CONN_KEYBD	PROBE
U73	74157	I/O MODULE
U83	74157	I/O MODULE
U84	74157	I/O MODULE
U65	74257	I/O MODULE
U66	74257	I/O MODULE
U33	7SEGLED	PROBE
U47	7SEGLED	PROBE
U61	7400	I/O MODULE
U70	7400	I/O MODULE
U71	7400	I/O MODULE
U62	7404	I/O MODULE

U59	74109	I/O MODULE
U80	7410	PROBE
U81	7410	PROBE
U7	74112	I/O MODULE
U25	74112	PROBE
U26	74125	I/O MODULE
U20	74148	I/O MODULE
U13	7414	I/O MODULE
U43	74164	I/O MODULE
U17	74164	I/O MODULE
U75	74175	I/O MODULE
U68	74244	I/O MODULE
U69	74244	I/O MODULE
U32	74244	I/O MODULE
U46	74244	I/O MODULE
U6	7430	I/O MODULE
U79	7430	I/O MODULE
U60	7431	I/O MODULE
U45	7432	I/O MODULE
U86	74373	I/O MODULE
U87	74373	I/O MODULE
U10	74373	I/O MODULE
U2	74374	I/O MODULE
U16	74374	I/O MODULE
U22	74374	I/O MODULE
U76	74374	I/O MODULE
U42	74390	I/O MODULE
U67	74590	I/O MODULE
U12	MAX232	PROBE
J4	PWRCONN	PROBE
U18	OSCILLATOR	PROBE
U82	74175	PROBE
U88	7486	PROBE
Y1	XTAL	PROBE
S4	KEYSWITCH	PROBE
S3	KEYSWITCH	PROBE
S2	KEYSWITCH	PROBE
S1	KEYSWITCH	PROBE
S6	KEYSWITCH	PROBE
DS1	LED	PROBE
Z1	NETWORK10	I/O MODULE

Appendix B
Demo/Trainer UUT Node List

NAME: NODELIST
DESCRIPTION: SIZE: 16,492 BYTES

```
U23-11  U41-2  U69-17  U30-11  U28-11  U41-14  Z1-10
U23-12  U40-2  U40-14  U69-15  U30-12  U28-12  Z1-9
U23-13  U39-2  U39-14  U69-13  U30-13  U28-13  Z1-8
U23-14  U38-2  U38-14  U69-11  U30-15  U28-15  Z1-7
U23-15  U37-2  U37-14  U69-8  U30-16  U28-16  Z1-6
U23-16  U36-2  U36-14  U69-6  U30-17  U28-17  Z1-5
U23-17  U35-2  U35-14  U69-4  U30-18  U28-18  Z1-4
U23-18  U34-2  U34-14  U69-2  U30-19  U28-19  Z1-3
U58-8 U34-15  U35-15  U36-15  U37-15  U38-15  U39-15  U40-15  U41-15
R69-1  R72-1  U88-8
U84-6  U72-32
R14-1  U46-12
R13-1  U46-14
R12-1  U46-16
R11-1  U46-18
R18-1  U46-3
R17-1  U46-5
R16-1  U46-7
R15-1  U46-9
U32-11 U31-40
R27-1  U32-9
R25-1  U32-12
R24-1  U32-14
R23-1  U32-16
R19-1  U32-18
R30-1  U32-3
U32-13  U31-39
R28-1  U32-7
U32-15  U31-38
```

```
R29-1  U32-5
U32-8  U31-1
R27-2  U33-7
R19-2  U33-1
U2-5  U66-6  U21-2  U30-26  U29-26  U28-26  U27-26
U84-3  U72-31
U84-10  U72-33
U16-15  U65-5  U84-11  U11-4  U72-38  U31-8  U30-9  U29-9  \
  U28-9  U27-9
U11-36  Y1-1   C8-1
U11-37  Y1-2   C9-1
U16-19  U61-9  U21-4  U62-9  U62-11
U70-11  U81-5
U22-9  U61-10  U57-13  U62-13
U65-1  U66-1  U60-7
U3-18  U48-2  U48-14  U68-2  U10-19  U11-21  U72-15  U31-27  \
  U29-19  U27-19
U2-6  U66-3  U21-1  U30-2  U29-2  U28-2  U27-2
R25-2  U33-8
R24-2  U33-10
R23-2  U33-13
R29-2  U33-11
U32-6  U31-2
U32-2  U31-4
U32-4  U31-3
U46-11  U31-22
R12-2  U47-13
R17-2  U47-11
R13-2  U47-10
R14-2  U47-8
R15-2  U47-7
R11-2  U47-1
U58-2  U8-14
U61-1  U62-12
U61-4  U62-10
U43-11  U61-12  U67-11  U67-13  U44-1  U44-13  U59-13
U61-6  U68-1  U68-19  U74-21
U61-3  U69-1  U69-19  U85-21
U70-3  U71-2
U70-6  U71-4
U70-8  U71-5
U56-10  U21-15  U72-2
U75-5  U83-10  U72-29
U68-3  U74-9  U77-6
U68-5  U74-10  U77-5
U68-14  U74-15  U77-24
U76-6  U78-4
U76-5  U78-36
U88-9  U78-29
U87-13  U77-16
```

```
U87-8  U77-15
U87-17  U77-18
U87-7  U77-13
U87-18  U77-19
U75-2  U77-10
U75-7  U77-9
U75-10  U77-8
U86-19  U78-19
U87-14  U77-17
U87-3  U77-11
U87-4  U77-12
U86-15  U78-17
U86-16  U78-16
U86-12  U78-25
U86-6  U78-18
U69-9  U86-13  U85-13
U69-18  U85-17  U77-26
U72-23  U78-11
U69-12  U86-8  U85-14
U80-8  U81-13
U80-10  U81-10  U82-6
U80-12  U81-1
U80-2  U70-5  U71-12  U81-4  U82-15
U80-4  U70-9  U81-11  U82-10
U80-6  U81-2
U80-11  U79-11  U82-14
U80-5  U79-4  U62-2
U80-3  U70-13  U79-5  U81-9  U82-3  U73-1  U83-1  U84-1  U62-5
U70-12  U76-11  U79-3  U86-11  U87-11  U72-16  U78-33
U71-13  U79-6  U81-3  U82-11
U22-5  U21-6
U83-9  U74-3  U85-3
U3-11  U55-2  U55-14  U68-17  U10-2  U11-28  U72-8  U31-34  \
  U29-11  U27-11
U32-17  U31-37
U2-15  U65-6  U73-11  U30-24  U29-24  U28-24  U27-24
U16-5  U66-5  U83-11  U30-5  U29-5  U28-5  U27-5
U46-13  U31-23
U46-8  U31-21
U2-9  U65-13  U30-23  U29-23  U28-23  U27-23
U27-22 U6-5  U45-3  U28-22
U34-4  U35-4  U36-4  U37-4  U38-4  U39-4  U40-4  U41-4  U48-4  \
  U49-4  U50-4  U51-4  U52-4  U53-4  U54-4  U55-4 U64-8  U63-8
U34-5  U35-5  U36-5  U65-4  U67-15 U37-5  U38-5  U39-5  U40-5  \
  U41-5  U48-5  U49-5  U50-5  U51-5  U52-5  U53-5 U54-5  U55-5
U34-6  U35-6  U36-6  U65-9  U67-2 U37-6  U38-6  U39-6  U40-6  \
  U41-6  U48-6  U49-6  U50-6  U51-6  U52-6  U53-6  U54-6  U55-6
U34-7  U35-7  U36-7  U65-7  U67-1  U37-7  U38-7  U39-7  U40-7  \
  U41-7  U48-7  U49-7  U50-7  U51-7  U52-7  U53-7  U54-7  U55-7
U34-3  U35-3  U36-3  U37-3  U38-3  U39-3  U40-3  U41-3  U48-3  \
```

```
  U49-3  U50-3  U51-3  U52-3  U53-3  U54-3  U26-8  U55-3
U34-12  U35-12  U36-12  U65-12  U67-3  U37-12  U38-12  U39-12  \
  U40-12  U41-12  U48-12  U49-12  U50-12  U51-12  U52-12  U53-12 \
  U54-12  U55-12
U34-11  U35-11  U36-11  U66-4  U67-4  U37-11  U38-11  U39-11  \
  U40-11 U41-11  U48-11  U49-11  U50-11  U51-11  U52-11  U53-11 \
  U54-11  U55-11
U34-10  U35-10  U36-10  U66-7  U67-5  U37-10  U38-10  U39-10  \
  U40-10  U41-10  U48-10  U49-10  U50-10  U51-10  U52-10  U53-10 \
  U54-10  U55-10
U34-13  U35-13  U36-13  U66-9  U67-6 \
  U37-13  U38-13  U39-13  U40-13  U41-13  U48-13  U49-13  U50-13   \
  U51-13  U52-13  U53-13  U54-13  U55-13
U58-11  U48-15  U49-15  U50-15  U51-15  U52-15  U53-15  U54-15   \
  U55-15
U6-2  U8-11
U6-3  U8-10
U5-10  U11-9  U72-3  U31-36  U15-11
U57-2  U5-13
U83-12  U74-4  U85-4
U6-11  U81-8
U16-2  U66-11  U83-5  U30-4  U29-4  U28-4  U27-4
U43-9  U56-12
U2-12  U65-10  U73-5  U30-21  U29-21  U28-21  U27-21
U56-1  U44-9  U64-12
U34-9  U35-9  U36-9  U66-12  U67-7  U37-9  U38-9  U39-9  U40-9  \
  U41-9  U48-9  U49-9  U50-9  U51-9  U52-9  U53-9 U54-9  U55-9
U56-9  U21-14  U11-39
U46-15  U31-24
U46-17  U31-25
U76-15  U78-38
U75-15  U77-7
U69-14  U86-7  U85-15
U45-1  U45-4  U56-3  U79-2  U57-1  U15-8
U45-5   U9-9  U30-20  U29-20
U2-16  U65-3  U73-14  U30-25  U29-25  U28-25  U27-25
U2-19  U66-14   U83-2  U30-3  U29-3  U28-3  U27-3
U46-6  U31-20
U46-4  U31-19
U46-2  U31-18
U16-6  U66-2 U83-14  U30-6  U29-6  U28-6  U27-6
U3-14  U52-2  U52-14  U68-11  U10-9  U11-19  U72-11  U31-31   \
  U29-15  U27-15
U3-13  U53-2  U53-14  U68-13  U10-6  U11-27  U72-10  U31-32   \
  U29-13  U27-13
U88-4  U78-28
U82-1  U13-10
U3-15  U51-2  U51-14  U68-8  U10-12  U11-26  U72-12  U31-30   \
  U29-16  U27-16
U22-4  J5-66  U14-66
```

```
U22-14  J5-13  U14-13
U22-18  J5-15  U14-15
U2-4  J5-17  U14-17
U22-13  J5-12  U14-12
U22-17  J5-14  U14-14
U14-52  C4-1
J5-52   C13-1
U16-8   J5-27  U14-27
U16-7   J5-26  U14-26
U16-13  J5-28  U14-28
U1-10  U17-8  U44-3  U44-11  U59-4  J5-31   \
  U7-1  U13-1  U14-31  U15-2
U16-14  J5-32  U14-32
U16-18. J5-34  U14-34
R1-2  U1-4  U19-1  J5-63  U4-12  U14-63  U15-1
U23-2  J5-51  U14-51
U23-3  J5-49  U14-49
U3-2  J5-50  U14-50
U3-6  J5-42  U14-42
U23-4  J5-47  U14-47
U23-5  J5-45  U14-45
U23-6  J5-43  U14-43
U23-8  J5-39  U14-39
U2-11  U16-11  U22-11  U7-2  U15-5
U56-5  U11-10  U72-1  U31-5  U15-12
U16-16  U65-2  U84-14  U11-2  U72-37  U31-9  U30-10  U29-10   \
  U28-10  U27-10
U23-9  J5-37  U14-37
U26-1  U13-4  U13-13  U14-64
U3-5  J5-44  U14-44
U22-8  J5-1  U14-1
U23-7  J5-41  U14-41
U3-9  J5-36  U14-36
J5-64  U13-12
U3-8  J5-38  U14-38
U3-7  J5-40  U14-40
U2-8  J5-19  U14-19
U2-2  U66-10  U21-3  U30-27  U29-27  U28-27  U27-27
U3-3  J5-48  U14-48
U2-18  J5-23  U14-23
U3-4   J5-46  U14-46
U84-12  U74-8  U85-8
U84-9  U74-7  U85-7
U1-16  J5-4  U14-4  U15-3
R26-1  U13-3
U16-12  U65-11  U84-5  U11-6  U72-39  U30-8  U29-8  U28-8   \
  U27-8
U6-4  U45-6  U30-22  U29-22
U21-13  U4-10  U31-6
U3-12  U54-2  U54-14  U68-15  U10-5  U11-18  U72-9  U31-33   \
```

```
  U29-12  U27-12
R5-1  S1-1  U31-14
R6-1  S2-1  U31-15
R7-1  S3-1  U31-16
R8-1  S4-1  U31-17
U56-8  U5-5
U68-7  U74-11  U77-4
U71-3 U82-4
U16-9  U65-14  U84-2  U11-7  U30-7  U29-7  U28-7  U27-7
U61-13 U58-6  U59-2  U59-3  U60-1  U63-9
U58-12  U62-8
U56-13  U59-12  U13-2
U65-15  U66-15 U44-5  U44-12
U2-7  J5-18  U14-18
U1-12  U19-3  J5-29  U11-38  U13-11  U31-35  U14-29
U1-15  J5-5  U14-5  U15-19
U84-4  U74-5  U85-5
U84-13  U72-34
U88-13  U72-18
U88-1  U72-19
U73-13  U72-26
U73-10  U72-25
U72-7  U78-8
U83-7  U74-2  U85-2
U83-4  U74-1  U85-1
U6-8  U5-1
U13-5  U13-8
R33-1  U20-4
U76-9  U78-37
R20-1  R21-1  R22-1  U12-2
U19-2  U7-3
U43-8  U42-3
U86-9  U78-14
U17-9  U4-11  U5-2
U7-5  U8-4
U19-4  U7-15
U45-9  U56-6
U84-7  U74-6  U85-6
U20-6  U10-7
U76-19  U63-4  U63-13
U20-2  U11-24  R3-1
U76-16  U63-5  U78-2
U69-7  U86-14  U85-11
U8-13  U62-1
U45-10  U5-8
U75-9  U62-4
U63-6  U78-39
U76-2  U63-12  U78-5
U80-9  U80-13  U70-1  U70-4  U70-10  U82-2
U81-12  U82-12
```

```
U68-16  U74-16  U77-21
U75-13  U83-3  U72-27
U68-12  U74-14  U77-25
U57-6  U5-12
U69-5  U86-17  U85-10
U69-16  U85-16  U77-2
J2-2  U12-7
J2-3  U12-13  R21-2
U1-13  U42-4
U25-1  U25-9  U78-32
U80-1  U61-2  U61-5  U70-2  U82-7
U71-9  U79-8
U60-2  U60-5  U60-15
U20-9  U10-3
U20-7  U10-4
U11-13  U12-10
R34-1  U25-15
U63-11  U78-6
U76-12  U78-3
U68-18  U74-17  U77-23
U69-3  U86-18  U85-9
U75-4  U83-13  U72-30
U75-12  U83-6  U72-28
U73-6  U72-24  U78-13
U4-5  U5-6
U4-1  U5-11
U11-35  U13-6
U11-5  U12-9
U60-14  U19-5
U44-2  U64-13
U11-14  U12-11
U4-2  U5-9  U15-13
U57-4  U9-5
U57-9  U15-16
U22-12  U57-3  U8-5
U20-12  U11-15  R2-1
U3-17  U49-2  U49-14  U68-4  U10-16  U11-25  U72-14  U31-28  \
  U29-18  U27-18
U68-9  U74-13  U77-3
U62-3  U72-17  U78-12
U22-6  U21-5  U8-6  U9-6
U12-1 C15-1
J6-2  U11-33  U13-9  R31-1  C7-1
U16-3  J5-24  U14-24
U16-17  J5-33  U14-33
R80-1  J5-61  U14-61
R77-1  J5-59  U14-59
U20-15  J5-57  U14-57
R78-2  J5-54  U14-54
U16-4  J5-25  U14-25
```

```
U1-5  U25-5
J5-16  U14-16 U2-3
U3-1  U23-1  U15-17
U1-2   U4-6
U26-2  U14-65
U45-8  U24-5  U4-13
U24-4  U19-6
U26-9  U56-4  U15-9
U1-11  R10-2  R9-2 C5-1  CR1-2
R22-2  J2-20
J2-5  U12-8  R20-2
J2-4  U12-14
U11-17  J6-3
U73-7  U74-19  U85-19
U62-6  U74-20  U85-20
U73-12  U74-23  U85-23
U73-9  U74-22  U85-22
U11-11  U12-12
U12-3 C15-2
U12-4 C17-1
U80-7
R10-1  S6-1
U3-19  U23-19  U57-8
U22-16  U8-2  U9-2
U22-15  U8-3  U9-3
U17-11  U5-4
U4-3  U4-9  U10-1  U10-11
U3-16  U50-2  U50-14  U68-6  U10-15  U11-20  U72-13   \
  U31-29  U29-17  U27-17
U64-9  U24-6
U6-6  U59-6
U56-11  U4-8
U61-8  U79-12
U61-11  U59-11
U57-5  U8-12
U56-2  U67-14  U44-6
U45-2   U9-7  U28-20  U27-20
U58-1  U8-15
U44-8  U63-10
U57-12  U58-10
U58-5  U59-9  U64-11
U58-9  U58-13  U64-10
U22-19  U66-13  U8-1  U9-1
U22-7  J5-67  U14-67  U15-18
U2-13  J5-20  U14-20
U2-14  J5-21  U14-21
U2-17  J5-22  U14-22
R79-2  J5-53  U14-53
U42-1  U42-7
U76-17  U87-16
```

```
U76-13  U87-12
U76-8  U87-9
U4-4  U5-3
U12-6 C16-2
U59-10  U59-14
U76-4  U87-5
U88-11  J3-9
U88-3  J3-8
R73-2  R71-2  J3-7
U12-5  C17-2
U18-8  U82-9  U25-13
U71-10  U71-11
U58-3  U58-4
R32-1  J6-1  C6-1
U76-14  U87-15
U76-3  U87-2
U71-6  U82-5
U71-8  U82-13
R67-2  Q2-1  Q1-2
U76-7  U87-6
R68-1  R70-1  U88-6
R28-2  U33-2
R30-2  U33-6
R16-2  U47-2
R18-2  U47-6
U71-1  U81-6
R70-2  R72-2  R66-2  Q2-2
R71-1  Q1-1
R61-1  R62-1  R63-1  R64-1  R65-1   U78-1
U76-18  U87-19
J2-7  R4-1
R35-1  DS1-2

! GROUND NODES

R73-1  U1-3  U1-9  U2-1  U2-10  U3-10  U6-7  U16-1   \
  U16-10  U22-1  U22-10 U23-10  U26-7  U26-10  U34-16   \
  U35-16  U36-16  U37-16  U38-16  U39-16  U40-16  U41-16  U43-7   \
  U45-7  U48-16  U49-16  U50-16  U51-16  U52-16  U53-16   \
  U54-16  U55-16  U56-7  U61-7  U65-8  U66-8  U67-8  U67-12   \
  U68-10  U69-10  U70-7  U71-7  U75-8  U76-1  U76-10   \
  U79-7  U81-7  U86-1  U86-3  U86-4  U86-10  U87-1  U87-10   \
  U88-2  U88-5  U88-7 U88-10  U17-7 \
  U42-2 U42-8  U42-12  U42-14  U42-15  U57-7  U58-7  U44-7  \
  U59-8  U82-8  U60-8  U73-8  U73-15  U83-8  U83-15  U84-8   \
  U84-15  U64-7  U24-7  U19-7  U20-5  U20-8  U21-8   \
  J5-9  J5-35  J5-60 U4-7  U5-7  \
  R4-2  U7-8  U8-8  U9-4  U9-8  U10-8   \
  U10-10  U10-13  U10-17  U10-18  U11-22  J3-1  J3-6   \
  U12-15  C16-1 U13-7 J4-6  J4-7  J4-8  J4-9  S4-2  S3-2   \
```

```
  S2-2  S1-2  S6-2 U25-8  C5-2  U32-1  U32-10  U32-19  U46-1   \
  U46-10  U46-19 Q2-3 U62-7 U63-7 U74-12 U74-18 J6-4 C4-2 C13-2 \
  U72-20  U85-12  U85-18  U77-14  U77-20  U77-22  U78-9   \
  U78-10  U78-15  U78-20  U78-21  U78-22  U78-23  U78-24  U78-31   \
  U31-7  U30-14  U29-14  U28-14  U27-14  U14-9  U14-35  U14-60  \
  U15-6  U15-7  U15-10  C1-2  C6-2  C7-2  \
  U18-7  R35-2   R77-2   R80-2 C8-2   C9-2   Z1-1

! POWER NODES

U18-1  DS1-1  R1-1  R34-2  R33-2  R3-2  U33-3  U33-14 R5-2  R6-2   \
  R7-2   R8-2   U80-14  R32-2  R31-2  R68-2  R69-2  R67-1   \
  R61-2  R62-2  R63-2  R64-2  R65-2  U1-1   U1-6   U1-17   \
  U1-18  U2-20  U3-20  U6-1  U6-12  U6-14  U16-20   \
  U22-3  U22-20  U23-20  U26-14  U34-8  U35-8  U36-8  U37-8   \
  U38-8  U39-8  U40-8  U41-8  U43-1  U43-2  U43-14   \
  U45-14  U47-3  U47-14  U48-8  U49-8  U50-8  U51-8  U52-8   \
  U53-8  U54-8  U55-8  U56-14  U61-14  U65-16  U66-16   \
  U67-10  U67-16  U68-20  U69-20  U70-14  U71-14  U75-1  U75-16   \
  U76-20  U79-1  U79-14  U81-14  U86-20  U87-20  U88-12   \
  U88-14  U17-1  U17-2  U17-14  R66-1  R79-1  R78-1   \
  R26-2  R9-1  J5-62  U14-62  U42-16  U57-14   \
  U58-14  U44-4  U44-10  U44-14  U59-1  U59-5  U59-15   \
  U59-16  U82-16  U60-6  U60-16  U73-16  U83-16  U84-16  U64-14   \
  U24-14  U19-14  U20-1  U20-3  U20-10  U20-11  U20-13   \
  U20-16  U21-16  J5-30  U4-14   U5-14 U7-4   \
  U7-16  U8-16  U9-16  U10-14  U10-20  U11-44  R2-2  U12-16   \
  U13-14  J4-10  J4-11  C1-1 J4-12  J4-13  U25-2   \
  U25-3  U25-4  U25-10  U25-11  U25-12  U25-14  U25-16  CR1-1   \
  U32-20  U46-20  Q1-3  U62-14  U63-14  U74-24   \
  J6-5  U72-36  U72-40  U85-24  U77-1  U77-27  U77-28  U78-7   \
  U78-30  U78-34  U78-35  U78-40  U31-26  U30-1  U30-28   \
  U29-1  U29-28  U28-1  U28-28  U27-1  U27-28  U14-30  U15-14   \
  U15-15  U15-20

! UNUSED OUTPUTS

U26-3
U73-4
U75-3
U75-6
U75-11
U75-14
U86-2
U86-5
U15-4
U59-7
U42-5
U42-6
U42-13
```

U42-11
U42-10
U42-9
U43-3
U43-4
U43-5
U43-6
U43-10
U43-12
U43-13
U67-9
U31-13
U31-12
U31-11
U31-10
U11-8
U11-40
U11-3
U11-43
U11-42
U11-41
U11-32
U11-31
U11-30
U11-16
U11-29
U8-9
U8-7
U9-15
U9-14
U9-13
U9-12
U9-11
U9-10
U21-12
U21-11
U21-10
U21-9
U21-7
U25-7
U25-6
U20-14
U17-3
U17-4
U17-5
U17-6
U17-10
U17-12
U17-13

```
*masters

! PROCESSOR ADDRESS LINES
U14-34
U14-33
U14-32
U14-28
U14-27
U14-26
U14-25
U14-24

U14-23
U14-22
U14-21
U14-20
U14-19
U14-18
U14-17
U14-16

U14-15
U14-14
U14-13
U14-12

! BUFFERED ADDRESS LINES
U16-19
U16-16
U16-15
U16-12
U16-9
U16-6
U16-5
U16-2

U2-19
U2-16
U2-15
U2-12
U2-9
U2-6
U2-5
U2-2

U22-19
U22-16
U22-15
U22-12
```

```
U22-9
U22-6
U22-5

! PROCESSOR DATA LINES

U14-51
U14-49
U14-47
U14-45
U14-43
U14-41
U14-39
U14-37

U14-50
U14-48
U14-46
U14-44
U14-42
U14-40
U14-38
U14-36

! BUFFERED DATA LINES
U23-18
U23-17
U23-16
U23-15
U23-14
U23-13
U23-12
U23-11

U3-18
U3-17
U3-16
U3-15
U3-14
U3-13
U3-12
U3-11
```

(This page is intentionally blank.)

Appendix C
Subprograms for Functional Test and Stimulus Programs

The following programs are included in this appendix:

abort_test
check_loop
check_meas
recover
tst_conten

```
program abort_test(ref)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! FUNCTIONAL TEST of the Microprocessor Bus.                               !
!                                                                         !
! This program is called by many of the test programs after the test     !
! program has found a failing circuit.  This program highlights the part!
! with the FAILED test attribute, changes all parts with a TESTING test !
! atribute to UNTESTED, and then checks to see if gfi has enough test    !
! results to make an accusation.  If an accusation exists then the       !
! accusation is displayed.  Otherwise a gfi hint is generated for the    !
! part and the test programs are terminated so that GFI can begin        !
! troubleshooting.                                                        !
!                                                                         !
! TEST PROGRAMS CALLED:                                                   !
!    none                                                                 !
!                                                                         !
! GRAPHICS PROGRAMS CALLED:                                               !
!    fail        (part_number)             Highlight part to be failed   !
!                                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare
   string ref                              ! The ref-pin of the failed part
   global numeric t2o                      ! Buffered I/O on /term2.
   global string array [1:107] part        ! Part shape and positions
   global numeric array [1:107] partatrb   ! Attribute number of part

   ! Next three items relate to Test window displayed by disply_pcb().

   global string testwind1 = "\1B[12;65f\1B[0m\1B[1m" ! Place text in line 2
   global string testwind2 = "\1B[13;65f\1B[0m\1B[1m" ! Place text in line 3
   global string undrtest = "\1B[15;66f\1B[0m"        ! Place text in line 5
end declare
```

```
! Highlight Failed Part.

   n = instr(ref, "-")
   if n = 0 then n = len(ref) + 1
   ic_num = (val(mid(ref, 2, n-2),16))

   ! convert decimal ic_num to hex

   dec100 = ic_num / $100
   dec10 = (ic_num - dec100 * $100) / $10
   dec1 = (ic_num - dec100 * $100 - dec10 * $10)
   hex_ic_num = dec100 * 100 + dec10 * 10 + dec1
   fail(hex_ic_num)

! Change all parts with a TESTING attribute to an UNTESTED attribute and
! display GFI TROUBLESHOOTING in the test window.

   for i = 1 to 107
      if partatrb[i] = 2 then untested(i)
   next
   print on t2o ,testwind1,"      GFI      ",testwind2,"TROUBLESHOOTING"
   print on t2o ,undrtest,"               "

! If GFI has an accusation then display the accusation otherwise generate
! GFI Hints.

   accusation = gfi accuse
   if accusation = "" then
      gfi hint ref
      fault 'gfi hints generated' '  please  run  gfi'
   else
      fault '' '' accusation
   end if

end program
```

```
program check_loop

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! This program checks the DEMO/TRAINER UUT Loopback switches.  If the    !
! loopback switches are not closed then a prompt is generated to close   !
! the loopback switches.  Otherwise no prompt is generated.              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

function pmpt_lpbk
   declare
      string q
   end declare

   print  "Close SW4-4, SW4-5 and SW6-4 for loopback"
   print  "Press \1B[7m ENTER \1B[0m key to continue "
   input  q  \  print
end function

  execute rs232_init()

  write addr $2006, data $AA
  wait time $200

  if ((read addr $2002) and $F) <> $D then
     execute pmpt_lpbk()
     return
  end if

  write addr $201E, data $FF
  write addr $2016, data $BB
  wait time $200

  if (read addr $2016) <> $BB then
     execute pmpt_lpbk()
     return
  end if

  write addr $201C, data $FF

  if ((read addr $201A) and 2) <> 0 then
     execute pmpt_lpbk()
     return
  end if

end program
```

```
program check_meas(dev, start, stop, clock, enable)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Check status of External START, STOP, CLOCK, ENABLE lines,                  !
! Return 1 if measurement is complete, display prompt to fix                  !
! the external lines, wait for ENTER key, and return 0 if the                 !
! measurement times out.                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare string dev
   declare string start
   declare string stop
   declare string clock
   declare string enable

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of program                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   times = 0
   loop while checkstatus(dev) <> $F and times < 100
      times = times + 1
   end loop

! If START fails then STOP, ENABLE and CLOCK will also fail.
! If ENABLE fails then CLOCK will also fail.
! Diagnose cause of failure and only display START if START fails.
! Do not display CLOCK when ENABLE line fails.

   if times < 100 then
      return (1)
   else
      t1 = open device "/term1", mode "unbuffered"

      ! turn autolinefeed off and clear screen

      print "\1B[2J\1B[201"
      n = checkstatus(dev) \ str = "" \ line = ""
      if (n and 4) = 0 then
         line = line + "START "
         str = str + " START to " + start + ","
      else
         if (n and 8) = 0 and stop <> "*" then
            line = line + "STOP, "
            str = str + " STOP to " + stop + ","
         end if
         if (n and 2) = 0 and enable <> "*" then
            line = line + "ENABLE "
            str = str + " ENABLE to " + enable + ","
         else if (n and 1) = 0 then
            line = line + "CLOCK "
            str = str + " CLOCK to " + clock + ","
         end if
      end if
      print "\1B[1;1f", "External line(s) ", line, "failed."
      print "\1B[2;1f", "Connect", str, "\1B[3;1f"
      print "Press \1B[7mENTER \1B[0m to REPEAT, \1B[7mNO \1B[0m to CONTINUE"

! Wait for ENTER key to be pressed.
```

```
      input on t1 ,str
      print "\1B[20h\1B[2J"
      close channel t1
      if str = "\7F" then
         return(1)
      else
         return (0)
      end if
   end if
end program
```

```
program recover

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! This program recovers sync between the 82288 Bus Controller and the  !
! 80286 pod.                                                           !
!                                                                      !
! Some of the stimulus programs disable ready before performing stimulus!
! which can cause the 80286 bus controller to get out of sync with the !
! pod.  The recover() program is executed to resynchronize the bus     !
! controller and the pod.                                              !
!                                                                      !
! TEST PROGRAMS CALLED:                                                !
!     (none)                                                           !
!                                                                      !
! GRAPHICS PROGRAMS CALLED:                                            !
!     (none)                                                           !
!                                                                      !
! Global Variables Modified:                                           !
!    recover_times                          Reset to Zero              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   declare global numeric recover_times   ! Count of executing recover().

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   Main part of STIMULUS PROGRAM                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   recover_times = recover_times + 1
   if recover_times <= 1 then
      podsetup 'enable ~ready' "off"        ! POD is out of sync with
      setspace(getspace("memory","word"))  ! the 82288 bus controller
      read addr 0                           ! Read in memory space then
      write addr 0, data 0                  ! Write in memory space to
      podsetup 'enable ~ready' "on"         ! synchronize 82288 and POD.
   else
      podsetup 'enable ~ready' "off"
      print "Please press the \1B[7mUUT RESET KEY \1B[0m"

      loop until (readstatus() and $10) <> 0   ! wait for RESET active.
      end loop
      podsetup 'enable ~ready' "on"

      loop until (readstatus() and $10) = 0    ! wait for RESET inactive.
      end loop
      print "\1B[2J"
   end if
end program
```

```
program tst_conten (addr, data_bits)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! TEST to isolate DATA BUS CONTENTION to the failing part.               !
!                                                                        !
! This program attempts to determine the cause of Data Bus contention by!
! testing the enable lines of all the devices on the Data Bus.  This    !
! program performs several steps.  First each device on the data bus is !
! accessed and determined to be accessible or inaccessible.  The        !
! variable bad_dev is a mask that records which devices failed.         !
!                                                                        !
! Many times when Data Bus contention exists, the device that has the   !
! bad enable lines can be accessed and the rest of the devices cannot be!
! accessed.  This program checks the mask to see if all except one      !
! device is bad and then tests the enable lines on the device that      !
! appeared good.                                                         !
!                                                                        !
! If all devices are bad or more than one device is good then this test !
! checks the enable lines of all the devices on the Data Bus by brute   !
! force.                                                                 !
!                                                                        !
! TEST PROGRAMS CALLED:                                                  !
!    abort_test  (ref-pin)                 If gfi has an accusation     !
!                                          display the accusation else  !
!                                          create a gfi hint for the    !
!                                          ref-pin and terminate the test!
!                                          program (GFI begins trouble- !
!                                          shooting).                   !
!                                                                        !
! FUNCTIONS CALLED:                                                      !
!    testic     (refname, pin1, pin2)     This function performs a gfi  !
!                                         test on refname.  Then the pins!
!                                         pin1 and pin2 (which are the  !
!                                         enable lines) are checked to  !
!                                         see if they are bad.  If so   !
!                                         abort_test is called and GFI is!
!                                         started on the failing enable !
!                                         line.  Otherwise all test info !
!                                         about the part is discarded   !
!                                         using the gfi clear command.  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Main Declarations                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declare
   numeric addr                          ! Address where failure occured.
   numeric data_bits                     ! Mask of failing data bits.
   numeric bad_dev = 0                   ! Mask to record failing devices
   numeric array [0:$15] ram_ic          ! Convert RAM bit to part number
   global string contention_checked      ! Record that this test ran.
end declare

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Functions                                                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

   function testic (ref, pin_a, pin_b)
      declare numeric ref
      declare numeric pin_a
      declare numeric pin_b
```

```
      ! convert decimal ref to hex

        dec100 = ref / 100
      dec10 = (ref - dec100 * 100) / 10
      dec1 = (ref - dec100 * 100 - dec10 * 10)
      href = dec100 * $100 + dec10 * $10 + dec1

      ref_a = "U" + str(href,16) + "-" + str(pin_a,16)
      ref_b = "U" + str(href,16) + "-" + str(pin_b,16)

      if gfi test ref_a fails then
         if (gfi status ref_a) = "bad" then
            abort_test(ref_a)
         else
            if (gfi status ref_b) = "untested" then gfi test ref_b
            if (gfi status ref_b) = "bad" then
               abort_test(ref_b)
            end if
         end if
         gfi clear       ! Only looking at Enable Lines, Clear Other Info.
      end if
   end function


   ram_ic[0] = 55   \  ram_ic[1] = 54          ! RAMs U55, U54
   ram_ic[2] = 53   \  ram_ic[3] = 52          ! RAMs U53, U52
   ram_ic[4] = 51   \  ram_ic[5] = 50          ! RAMs U51, U50
   ram_ic[6] = 49   \  ram_ic[7] = 48          ! RAMs U49, U48
   ram_ic[8] = 41   \  ram_ic[9] = 40          ! RAMs U41, U40
   ram_ic[10] = 39  \  ram_ic[11] = 38         ! RAMs U39, U38
   ram_ic[12] = 37  \  ram_ic[13] = 36         ! RAMs U37, U36
   ram_ic[14] = 35  \  ram_ic[15] = 34         ! RAMs U35, U34

   if contention_checked <> "yes" then
      contention_checked = "yes"
      podsetup 'report intr' "off"
      podsetup 'enable ~ready' "on"
      print "\nl\nlTESTING BUS CONTENTION"


! Read from each device on the bus and record if each device reads correctly.
!
! Then check and see if all components are bad except one.  If so then check
! that component's enable lines.
!
! Otherwise brute force check all enable lines on all components connected to
! the bus.

   ! ROM0 and ROM1

      setspace( getspace("memory", "word"))
      if (read addr $E002A) <> 0 then bad_dev = bad_dev or 1
      if (read addr $F0022) <> 0 then bad_dev = bad_dev or 2

   ! Dynamic RAM

      write addr $1000, data $FFFF
      if (read addr $1000) <> $FFFF then bad_dev = bad_dev or 4
      write addr $1000, data 0
      if (read addr $1000) <> 0 then bad_dev = bad_dev or 4

   ! PIA registers
```

```
      execute pia_init()
      if (read addr $4002) <> $FF then bad_dev = bad_dev or 8
      write addr $4002, data 0
      if (read addr $4002) <> 0 then bad_dev = bad_dev or 8

   ! DUART registers

      execute rs232_init()
      if (read addr $200A) <> $11 then bad_dev = bad_dev or $10
      if (read addr $201A) <> $FF then bad_dev = bad_dev or $10
      if (read addr $2012) <>  $C then bad_dev = bad_dev or $10

   ! Video Controller registers

      execute rs232_init()
      if (read addr 8) <> $FF then bad_dev = bad_dev or $20
      if (read addr $A) <>  0 then bad_dev = bad_dev or $20

! If only one device is good, CLIP and check enable lines on that device.

      if bad_dev <> 0 and bad_dev <> $3F then
      ! CLIP and Check Enable lines on ROMs
         if bad_dev = $7E then
            if (data_bits and $FF) <> 0 then        ! Low data bits are bad
               testic(29, $20, $22)                 ! Check low byte ROM0.
            end if
            if (data_bits and $FF00) <> 0 then      ! High data bits are bad
               testic(30, $20, $22)                 ! Check high byte ROM0.
            end if

         else if bad_dev = $7D then
            if (data_bits and $FF) <> 0 then        ! Low data bits are bad
               testic(27, $20, $22)                 ! Check low byte ROM0.
            end if
            if (data_bits and $FF00) <> 0 then      ! High data bits are bad
               testic(28, $20, $22)                 ! Check high byte ROM0.
            end if

         else if bad_dev = $7B then
            testic (ram_ic[msb(data_bits)], $15, 4) ! Check RAM.
         else if bad_dev = $77 then
            testic (31, 6, 6)                       ! Check PIA.
         else if bad_dev = $2F then
            testic (11, $39, 9)                     ! Check DUART.
         else if bad_dev = $1F then
            testic (72, 2, 3)                       ! Check Video Controller
         end if
      end if

! BRUTE FORCE check enable lines of all devices on bus.

      if (data_bits and $FF) <> 0 then              ! Low data bits are bad
         testic(27, $20, $22)                       ! Check low byte ROM0.
         testic(29, $20, $22)
      end if
      if (data_bits and $FF00) <> 0 then            ! High data bits are bad
         testic(28, $20, $22)                       ! Check high byte ROM0.
         testic(30, $20, $22)
      end if
      testic (ram_ic[msb(data_bits)], $15, 4)       ! Check RAM.
      testic (31, 6, 6)                             ! Check PIA.
      testic (11, $39, 9)                           ! Check DUART.
      testic (72, 2, 3)                             ! Check Video Controller
```

```
      testic (10, $11, 1)                          ! Check Interrupt Buffer
      if bad_dev = $3F then
         if (data_bits and $FF) <> 0 then
            if gfi test "U3-1" fails then abort_test("U3-1")
         end if
         if (data_bits and $FF00) <> 0 then
            if gfi test "U23-1" fails then abort_test("U23-1")
         end if
      end if

      print "BUS CONTENTION TEST PASSES"
   end if
end program
```

(This page is intentionally blank.)

Appendix D
Demo/Trainer UUT Schematics

Index