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 x $2^9 + 2^8$ 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

STIMULUS	MEASUREMENT
POD 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

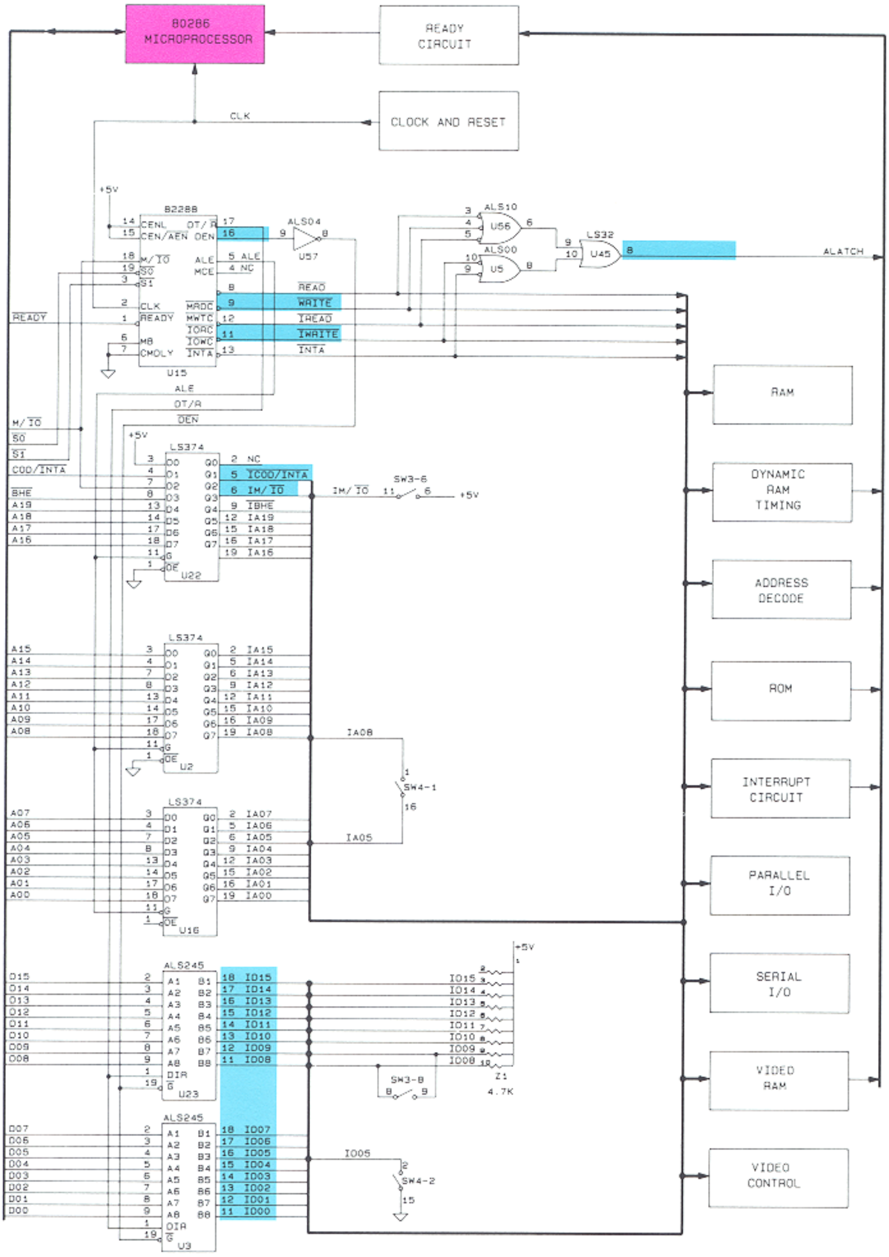

Figure 4-96: Bus Buffer Functional Test (Part A)

Keystroke Functional Test (Part B)

CONNECTION TABLE

STIMULUS	MEASUREMENT
POD	I/O MOD
TEST ACCESS SOCKET	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

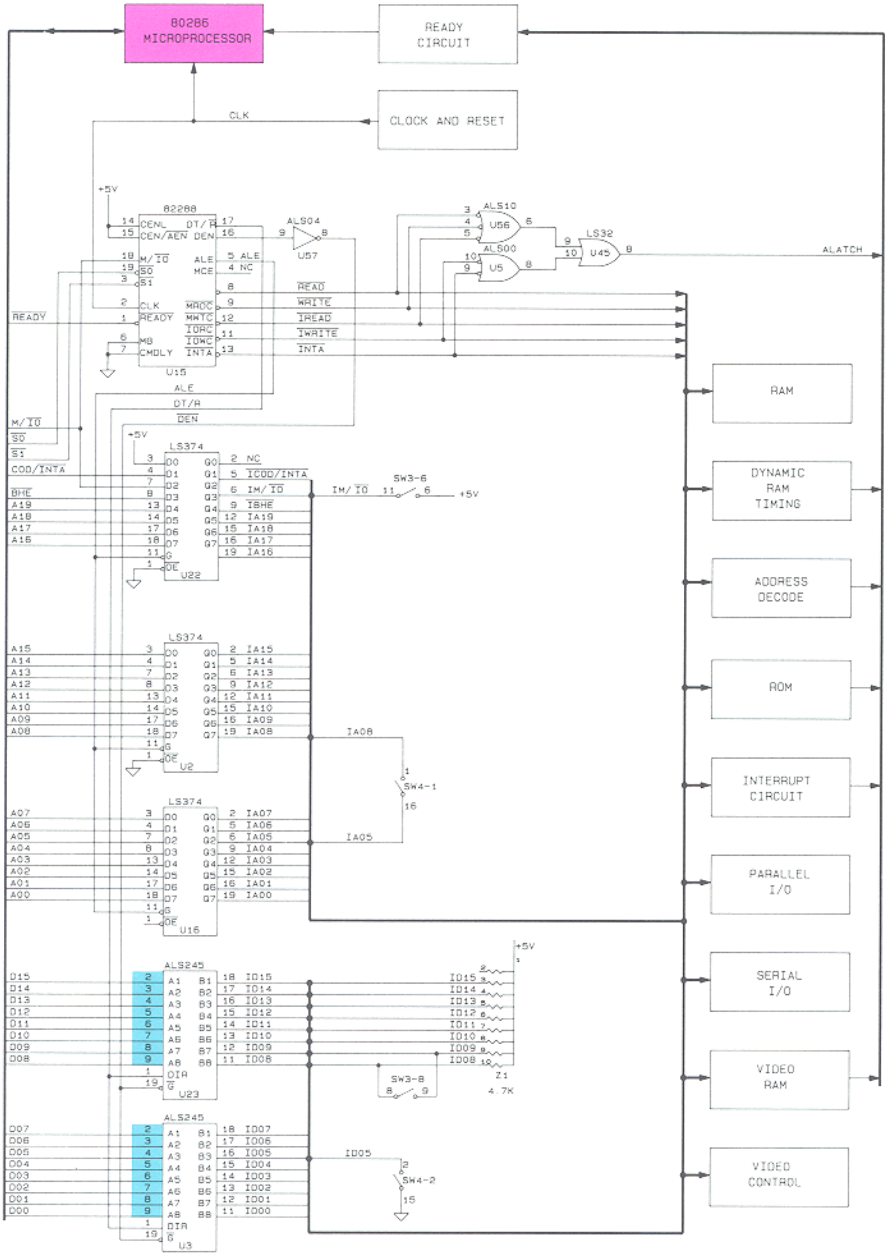

Figure 4-97: Bus Buffer Functional Test (Part B)

Keystroke Functional Test (Part C)

CONNECTION TABLE

STIMULUS	MEASUREMENT
POD 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	D62F	
IA01	-16	16	B21A	
IA02	-15	15	DA07	
IA03	-12	12	1024	
IA04	-9	9	E030	
IA05	-6	6	4442	
IA06	-5	5	4F2A	
IA07	-2	2	0772	
IA08	U2-19	19	9635	
IA09	-16	16	1734	
IA10	-15	15	41BA	
IA11	-12	12	455A	
IA12	-9	9	BEED	
IA13	-6	6	3868	
IA14	-5	5	DEE6	
IA15	-2	2	C265	
IA16	U22-19	19	F8A7	
IA17	-16	16	9196	
IA18	-15	15	25E4	
A19	-12	12	4A74	
IBHE	-9	9	AC5F	
ALE	U15-5	5	2A42	1 0
INTA	-13	5	2A42	1
DT/R	-17	17	9D92	1 0
ALATCH	U45-8	14	B7D0	1 0

RESPONSE TABLE #2

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
READ	U15-8	8	4513
IREAD	-12	12	6F51

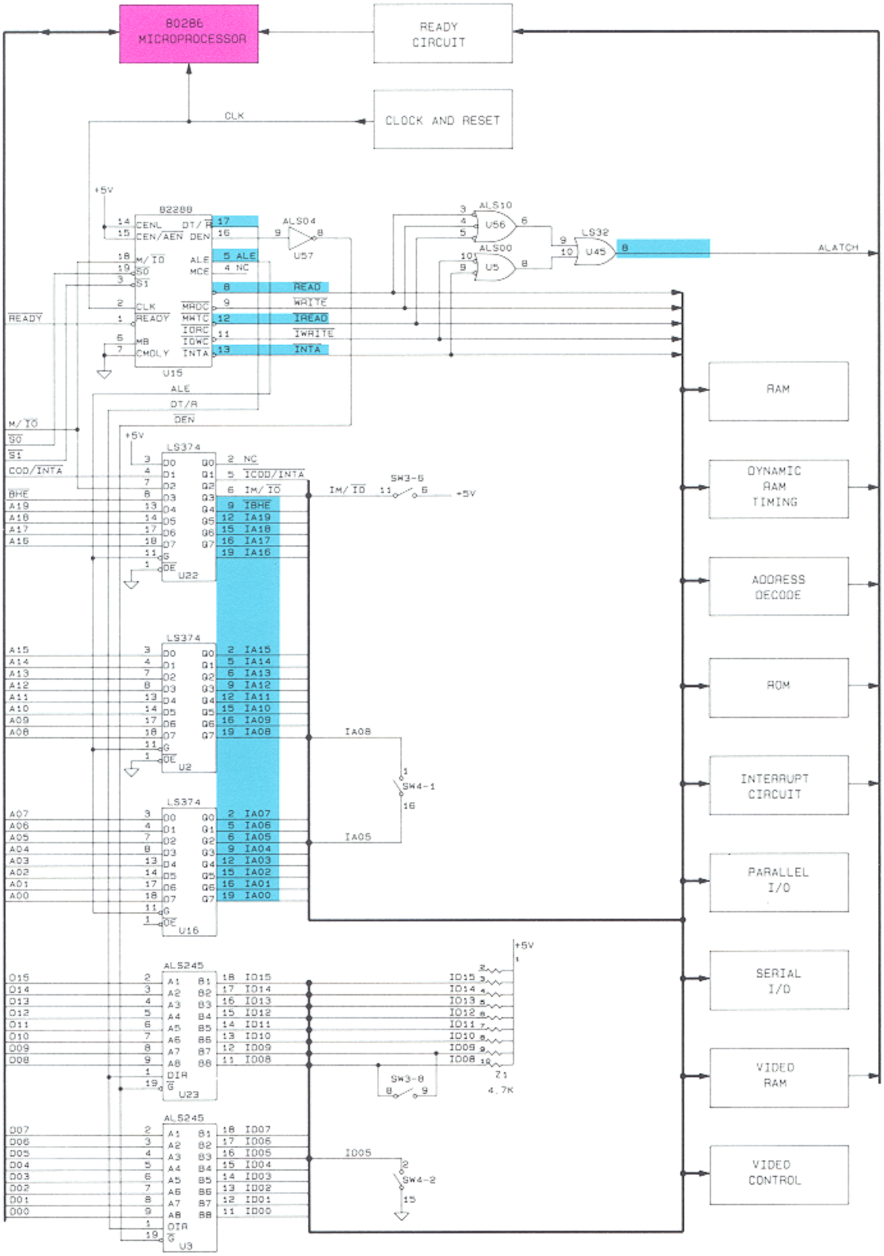

Figure 4-98: Bus Buffer Functional Test (Part C)

Keystroke Functional Test (Part D)

CONNECTION TABLE

STIMULUS	MEASUREMENT
POD 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

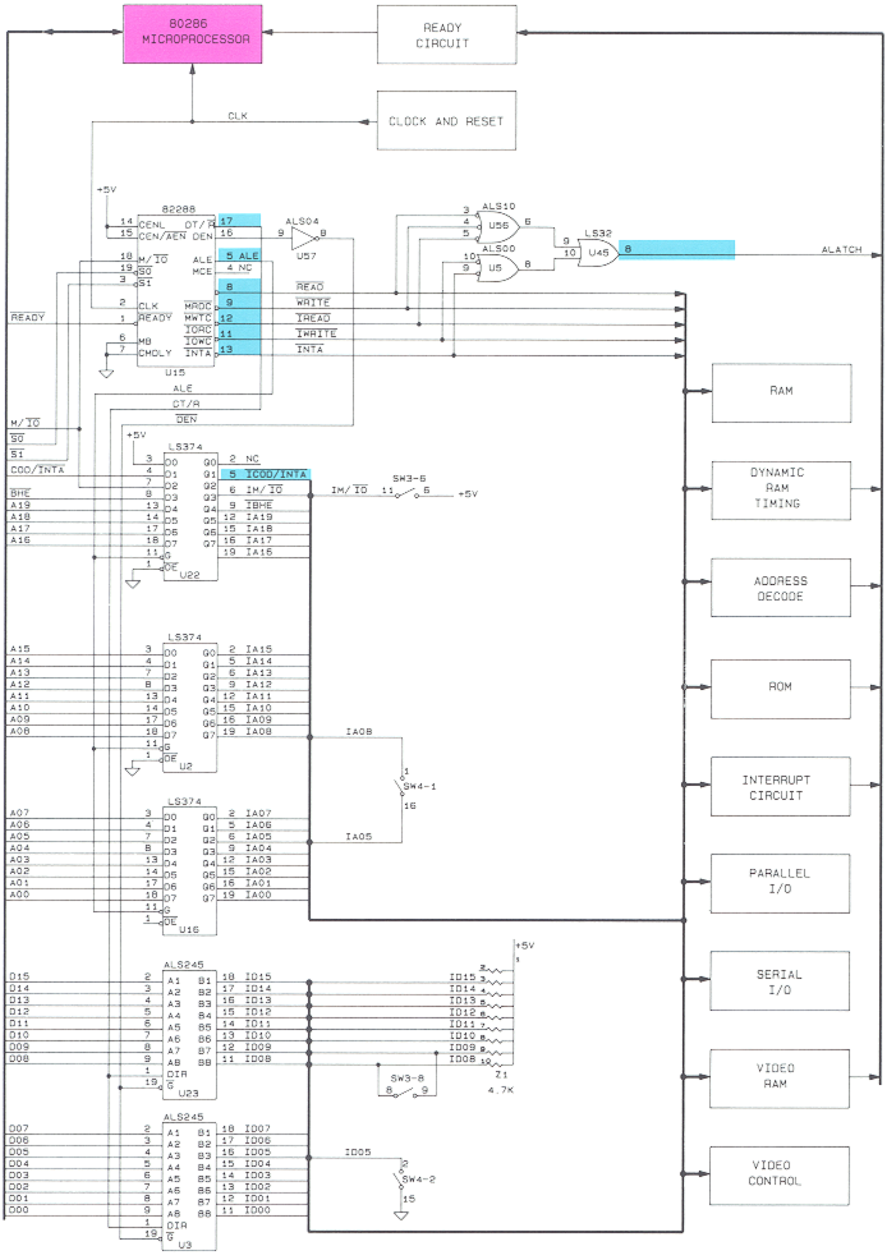

Figure 4-99: Bus Buffer Functional Test (Part D)

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 "\nlTESTING 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

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

PROGRAM: CTRL_OUT1

EXERCISES CONTROL LINES FROM THE MICROPROCESSOR USING POD ADDRESS SYNCHRONIZATION

MEASUREMENT AT:

U22-5,6
U57-8
U15-16
U45-8

PROGRAM: CTRL_OUT2

EXERCISES CONTROL LINES FROM THE MICROPROCESSOR USING DATA SYNCHRONIZATION

MEASUREMENT AT:

U15-17,8,9,12,11,13,5
U56-6
U45-8
U5-8

PROGRAM: CTRL_OUT3

EXERCISES CONTROL LINES FROM THE MICROPROCESSOR USING INTERRUPT ACKNOWLEDGE SYNCHRONIZATION

MEASUREMENT AT:

U15-17,5,8,9,12,11,13
U56-6
U45-8
U5-8

PROGRAM: ROM1_DATA

EXERCISES ALL DATA LINES AS INPUTS TO THE MICROPROCESSOR

MEASUREMENT AT:

U3-9,8,7,6,5,4,3,2
U23-9,8,7,6,5,4,3,2

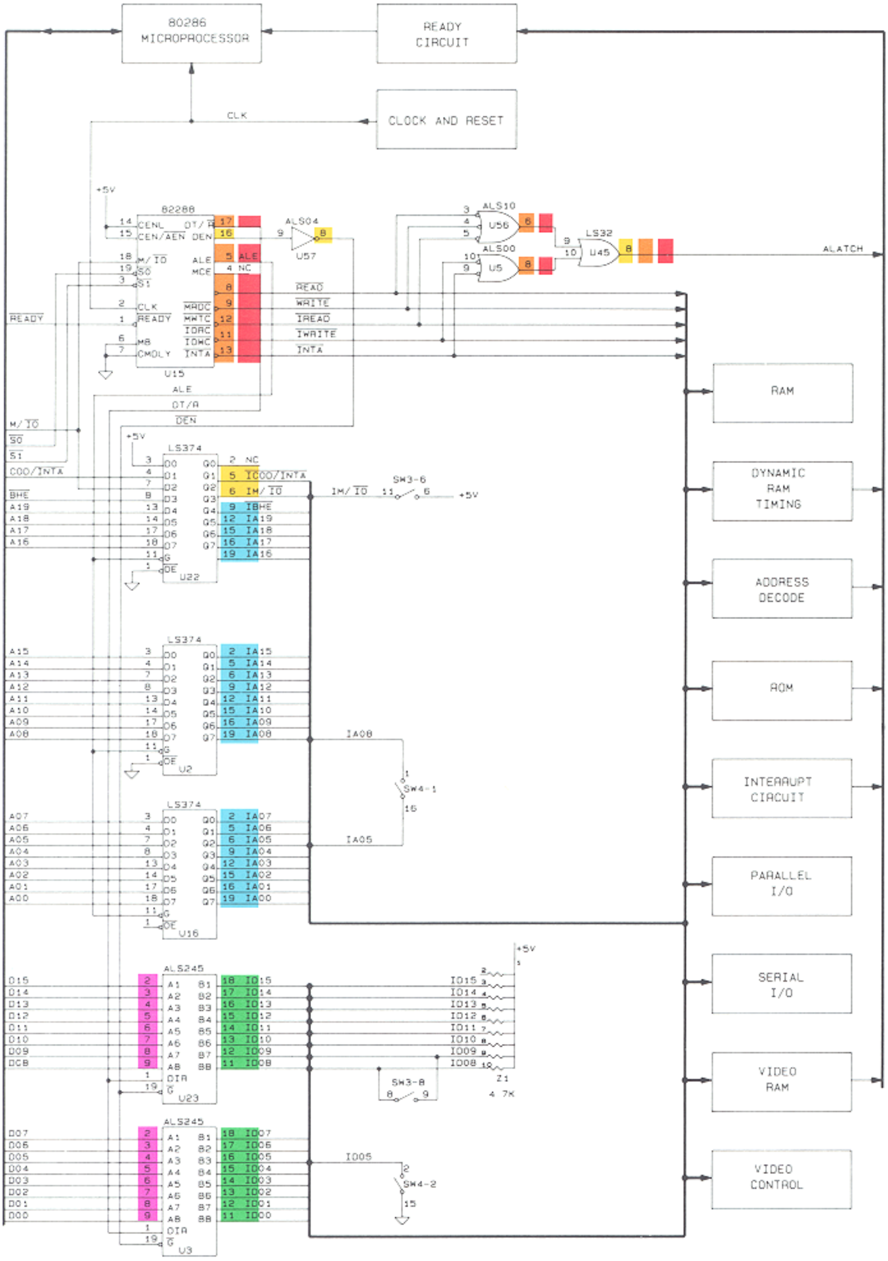

Figure 4-100: Bus Buffer Stimulus Program Planning

```
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

            ------------------- 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       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).

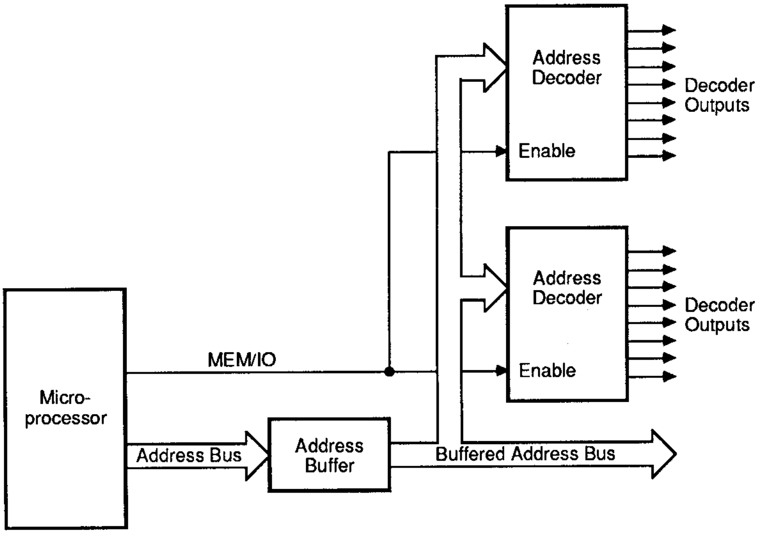

Figure 4-105: Typical Address Decode Functional Block

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

STIMULUS	MEASUREMENT
POD	I/O MOD
TEST ACCESS SOCKET	U8 U9 U21

RESPONSE TABLE

SIGNAL	PART PIN	I/O MOD PIN	SIGNATURE
RAM0	U8-15	19	9CA8
RAM1	U8-14	18	6BAD
VRAM	U8-13	17	902F
IPOLL	U8-12	16	6DEE
SPARE1	U8-11	15	930E
SPARE2	U8-10	14	6C7E
ROM0	U9-9	13	3C7B
ROM1	U9-7	7	3B4C
VIDSLT	U21-15	19	F8BE
I/OSLT	U21-14	18	092A
PPISLT	U21-13	17	354F

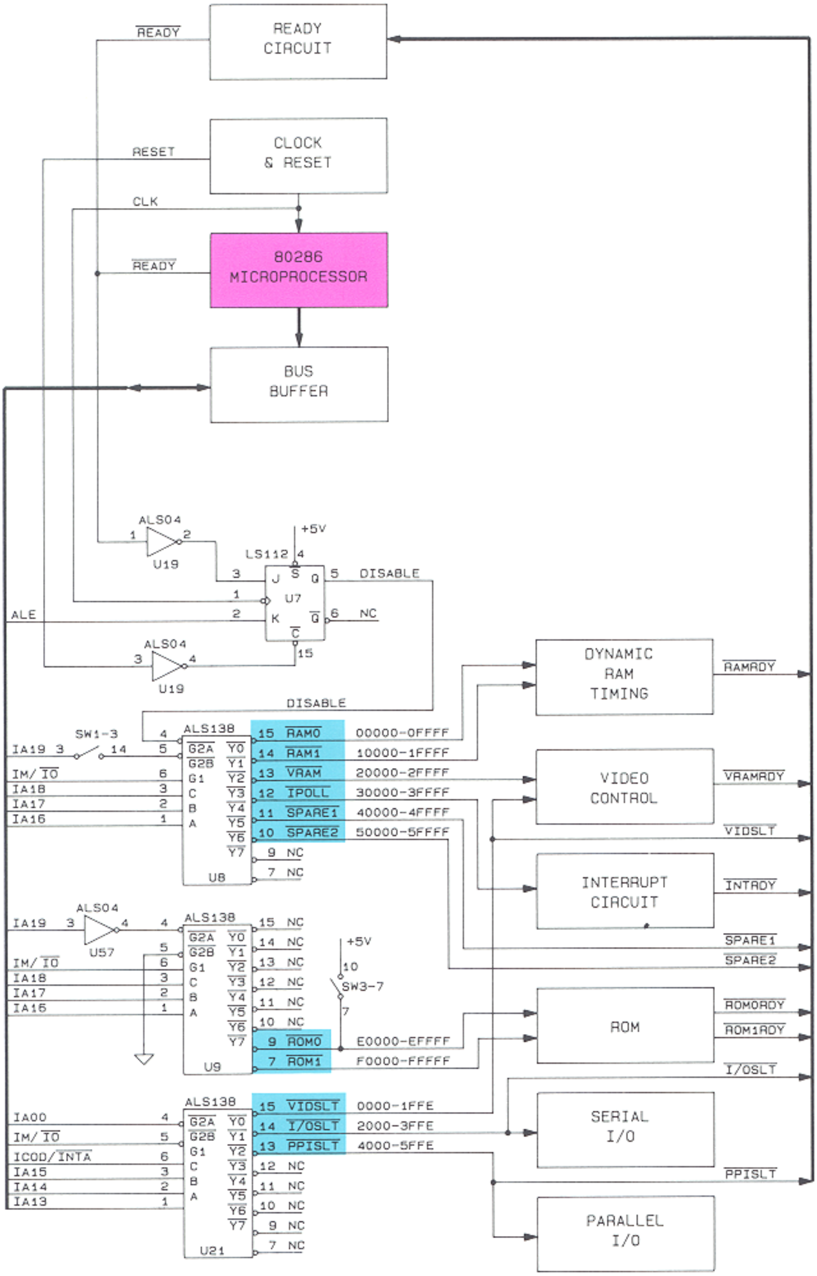

Figure 4-106: Address Decode Functional Test

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

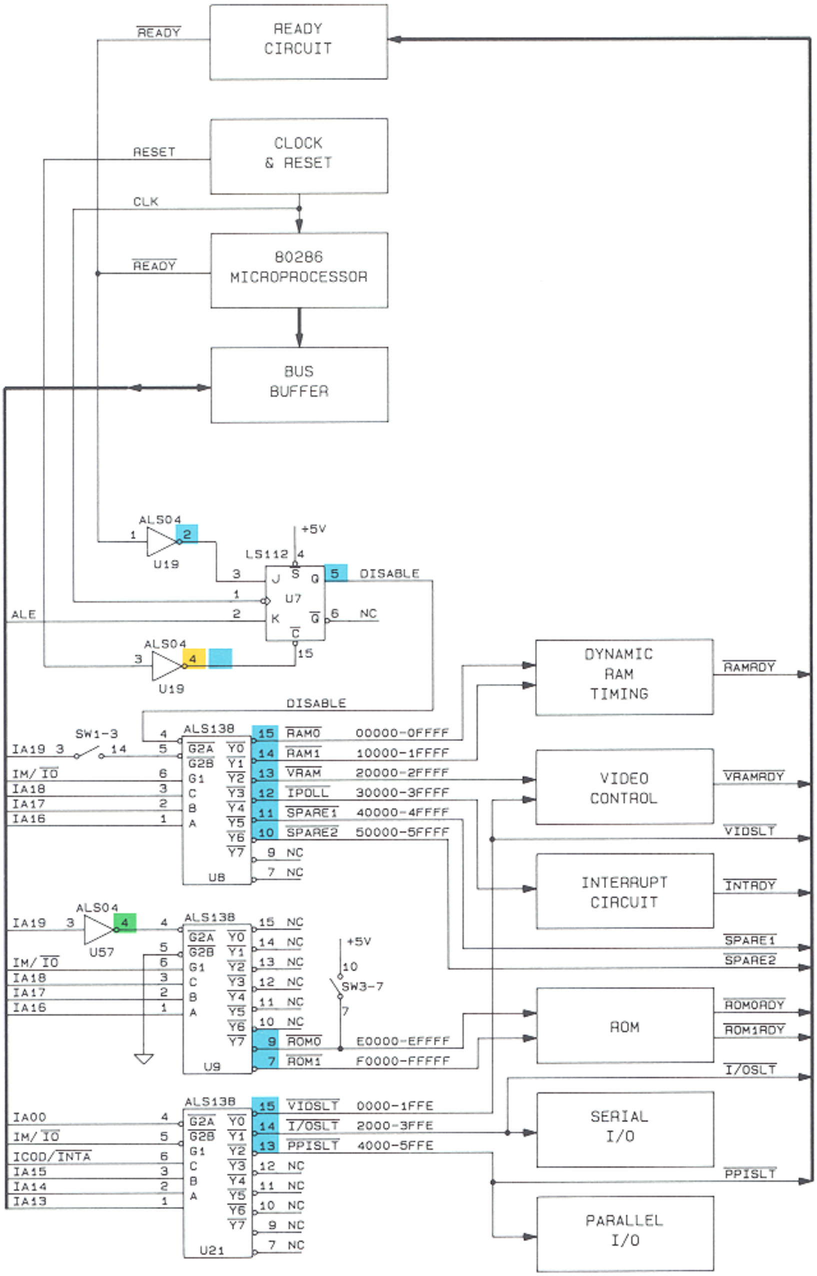

Figure 4-107: Address Decode Stimulus Program Planning

```
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

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

STIMULUS	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

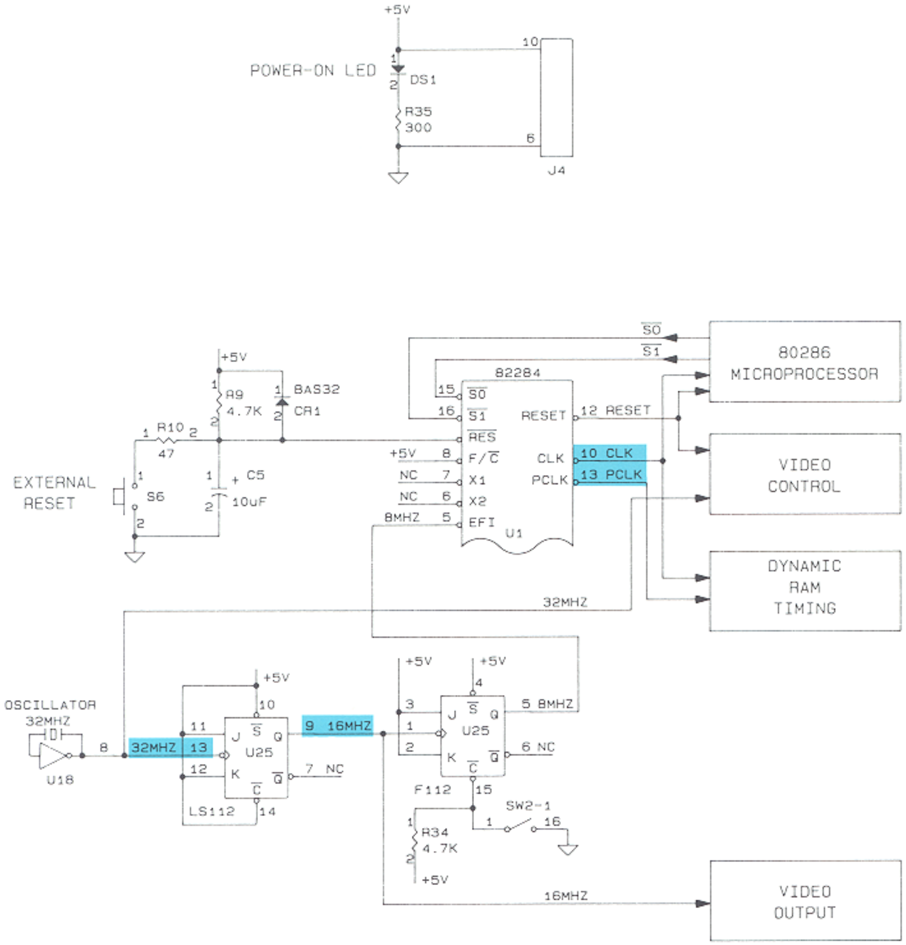

Figure 4-110: Clock and Reset Functional Test (Part A)

Keystroke Functional Test (Part B)

CONNECTION TABLE

STIMULUS	MEASUREMENT
RESET SWITCH S6	PROBE U1-12

STIMULUS AND RESPONSE TABLE

RESET SWITCH	PART PIN	LEVEL
RELEASED	U1-12	LOW
PRESSED	U1-12	HIGH

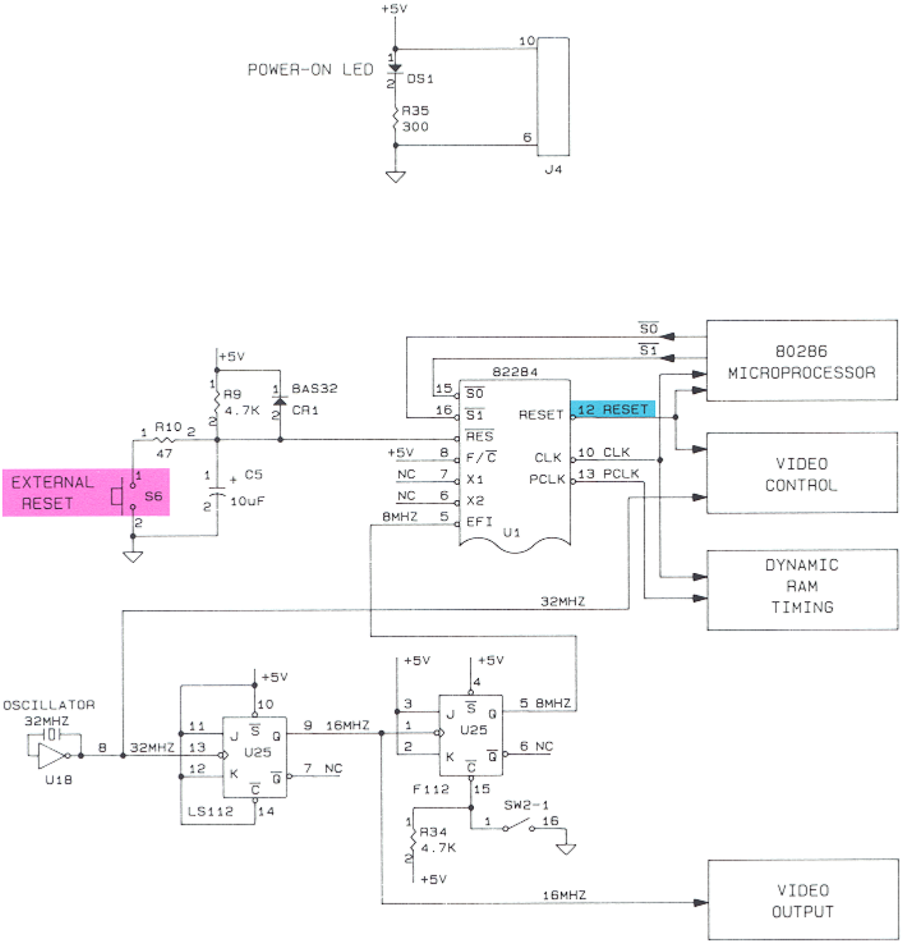

Figure 4-111: Clock and Reset Functional Test (Part B)

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

PROGRAM: LEVELS

MEASURES STATIC LEVELS

MEASUREMENT AT:

J4-10,6
R34-1
DS1-2

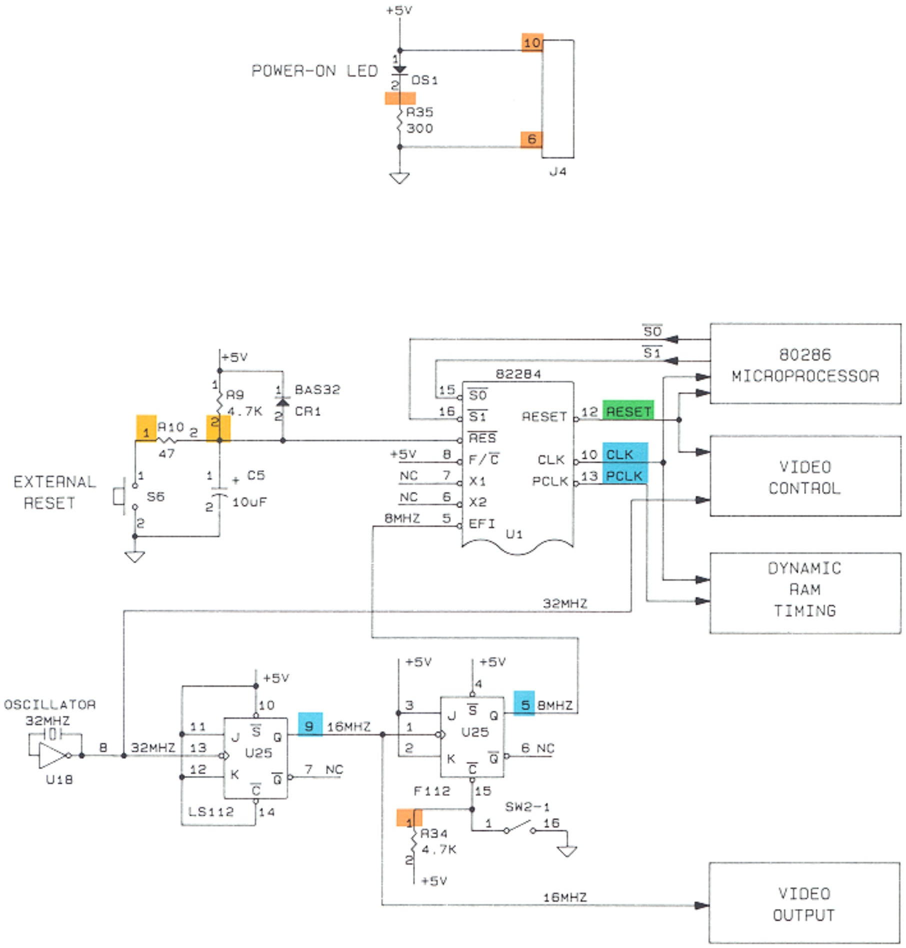

Figure 4-112: Clock and Reset Stimulus Program Planning

```
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 "\nl\nl"

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 "\nl\nl"

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

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

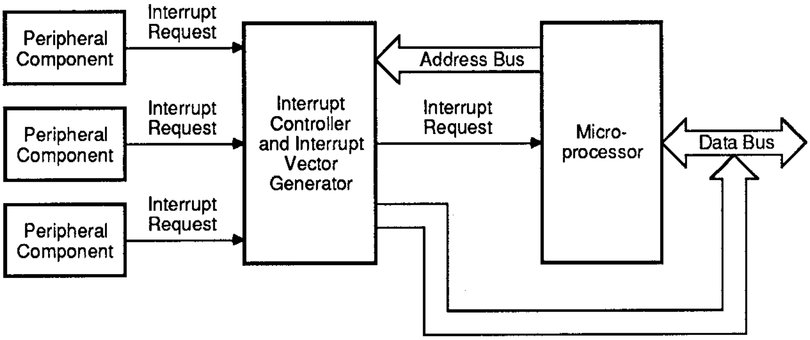

Figure 4-119: Typical Interrupt Circuit

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

STIMULUS	MEASUREMENT
POD TEST ACCESS SOCKET	POD TEST ACCESS SOCKET

STIMULUS AND MEASUREMENT TABLE

STIMULUS	MEASUREMENT	
	STATUS	ADDRESS 30000
NO INTERRUPTS	C0	27
TIMER INTERRUPT	C8	25
SERIAL INTERRUPT	C8	22

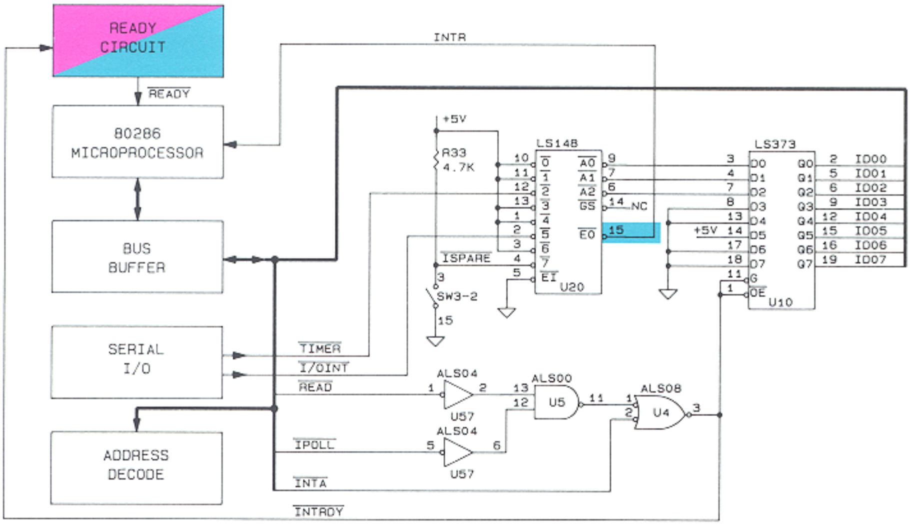

Figure 4-120: Interrupt Circuit Functional Test

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

PROGRAM: INTERRUPT

EXECUTES RS232_INIT AND EXERCISES INTERRUPT LINES

MEASUREMENT AT:

U10-2,5,6,9,12,15,16,19
U20-6,7,9,15
R33-1
U5-11

PROGRAM: DECODE

PERFORMS AN ACCESS FOR EACH DECODED BLOCK

MEASUREMENT AT:

U57-2,6
U5-11
U4-3

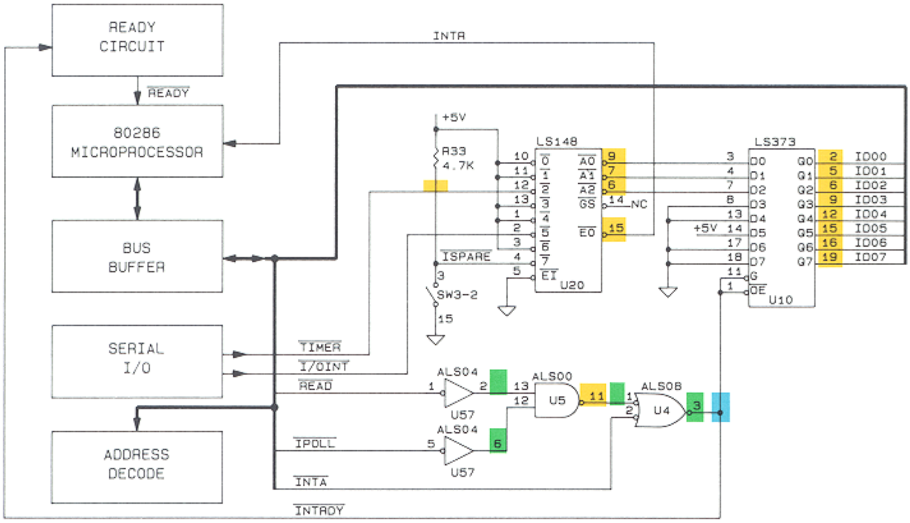

Figure 4-121: Interrupt Circuit Stimulus Program Planning

```
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

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

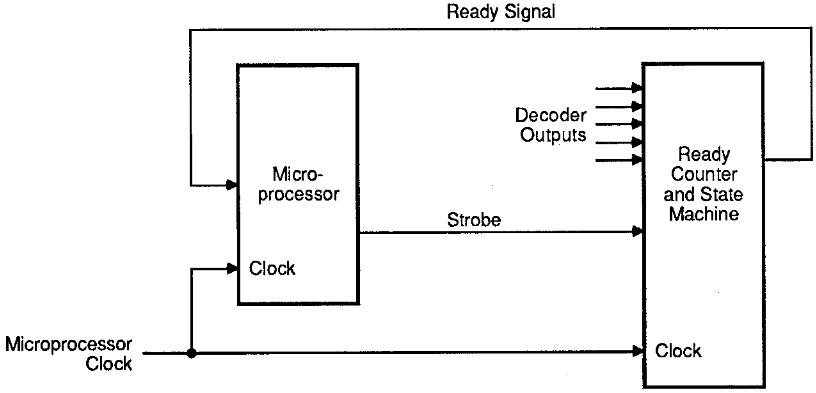

Figure 4-124: Typical Ready Circuit

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
```