APPLICATION NOTE 292
DATA DOMAIN MEASUREMENT SERIES

Minicomputer analysis techniques using Logic Analyzers

HEWLETT PACKARD

TABLE OF CONTENTS

INTRODUCTION

Minicomputer System designers and users can benefit from a better acquaintance with debugging and troubleshooting techniques which use logic analyzers. Logic analyzers are also valuable tools in software evaluation and code optimization. A quick way to get an overview of what a logic analyzer is and how it is used with minicomputers is to look at some common applications and measurements. While the focus is on logic analyzers and minicomputers, the techniques described are equally applicable to similar problems in microprocessors and mainframes.

SOFTWARE DIAGNOSTIC TECHNIQUES

Problem solving in a minicomputer system is not a simple task. A typical system may consist of a computer, several peripherals from various manufacturers or in-house design, and often, at least one intelligent peripheral built around one or more microprocessors. As the systems have evolved, a set of programs have been developed concurrently which are known, as a class, as software diagnostics. Software diagnostics are used to troubleshoot the central processing unit (CPU) and devices on memory and I/O buses. Well-designed software diagnostics will usually identify a faulty module, but not necessarily pinpoint the cause.

There are several major limitations to troubleshooting solely with software diagnostics. To run a diagnostic program, the system must be operating at a minimal level; a hung-up bus or a dead system can't be analyzed with software. Diagnostics often will not identify intermittent problems or errors in the application software. For a full set of troubleshooting tools, software diagnostics should be supplemented with instruments which monitor system lines, activity, and timing in real time. The complexities of troubleshooting minicomputer systems require the real-time, analytical capabilities of a logic analyzer.

LOGIC ANALYZERS

Today's microprocessor controlled logic analyzers (figure 1) provide a powerful measurement set capable of testing complex minicomputer systems. To be useful in troubleshooting minicomputer systems, the logic analyzer must be able to collect, store, and display information, perform some functional measurements, and interact with other measurement devices.

Figure 1. The powerful measurement set of today's logic analyzer is made possible by microprocessor control.

DATA ACQUISITION

Logic analyzers monitor minicomputer systems which are running at normal operational speeds. Two questions arise: how is data acquired, and what data is acquired?

Data Channels. A typical minicomputer has 16 or 18 bits of address, 16 bits of data, and control lines. For analysis of state flow, it is necessary to have simultaneous, correlated information from these inputs. Consequently, even early models of logic analyzers accommodate a minimum of 16 input channels plus 2 or more inputs from control lines or external signals, and more recently developed logic analyzers have 32 input channels with four or more auxiliary input channels.

Trace Specification. The data collected by a logic analyzer and shown on the display is called a trace. This trace is a window into the state flow of the minicomputer, and the process of determining where to place the window is called trace specification. The simplest trace specification is a **single-pattern trigger.** For simple, in-line program flow, a particular state is set on the analyzer, and when that state appears for the first time, it acts as a trigger for the logic analyzer memory and the subsequent states are stored.

Programs in minicomputer systems are rarely simple, and a more sophisticated trace specification is needed. Consider a program with many complex branching networks, (figure 2). A **sequential trigger,** as shown in figure 2, permits a trace to be made of a unique program path selected from several alternative paths. Sequential triggering can also be used to pick up parameters entered early in program flow for use in subsequent subroutines, and the logic analyzer displays these parameters first, just ahead of the trace listing of the subroutine.

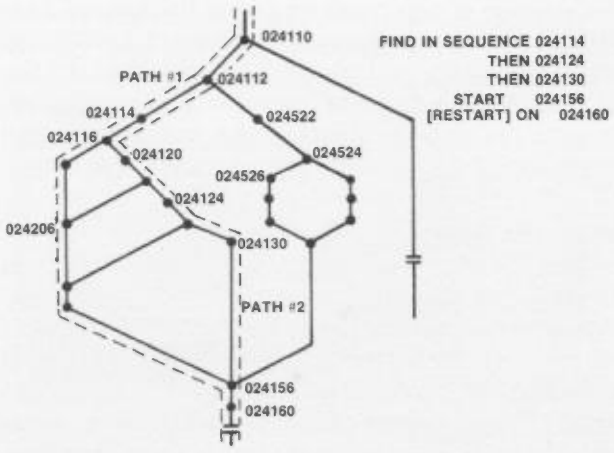


Figure 2. A sequential trace specification is needed to trace a specific path (#2) in a program with complex network branching.

A **multiple-occurrence trigger** is a valuable trace specification for unraveling nested loops in program flow. For example, a program may sequentially sample several peripherals (major loop), receive data from each peripheral (minor loop), and then format data from each transmission (subminor loop). The combination of a sequential trigger and a multiple-occurrence trigger could focus the analyzer window to view only the sixth data format of the third data transmission from peripheral three.

Two trace specifications which facilitate monitoring clock and control lines are the **ORed trigger** and the **ANDed trigger.** One logic analyzer (HP Model 1615A) has a **glitch trigger** which initiates a trace if an unwanted interference of specified magnitude (glitch) occurs on a designated signal line.

These trace specifications make analysis of minicomputer systems quicker and more effective. They allow the user to check only state flow that is pertinent, and to focus more quickly on software or hardware that may be malfunctioning.

DATA STORAGE

The result of trace specification is a selective trace, and only the states of interest are stored in the logic analyzer. Data captured for analysis can be restricted to a particular operation, specific activities, or a selected range of addresses. To capture every state activity would require an enormous memory capacity, and severely tax the analyst's time by forcing him to study reams of output to locate isolated malfunctions or inconsistencies. A selective trace provides the data needed and omits activities that are extraneous to the situation.

DATA DISPLAY

Once the data has been acquired and stored, it must be displayed in a meaningful format for analysis. Logic analyzers offer a variety of formats, which fall into three general categories: list displays, timing diagrams, and system overviews.

List Display

The most frequently used format is the list display (figure 3), which is a convenient format for detailed analyses of software execution. Freedom to specify the format varies with the logic analyzer used, but with the two models used for most examples in this paper, HP Model 1610A and HP Model 1615A, the operator can choose clock slopes, logic polarity, numerical bases, and group inputs under different labels.

Timing Diagrams

Model 1615A Logic Analyzer provides an 8-channel timing diagram (figure 4) as a display mode. Timing diagrams show the functional time relation between control lines, and these diagrams facilitate investigation of handshake and control state problems. The 1615A can display timing phenomena that occur prior to a trigger point, make single-shot multichannel measurements, and detect and trigger on glitches.

System Overview

A convenient display feature available on Model 1610A Logic Analyzer is a graph mode. A selected parameter, e.g. address, is plotted with magnitude on the Y-axis and sequence of occurrence on the X-axis (figure 5). With this form of display, an operator can quickly identify irregularities and discontinuities in program flow.

Figure 3. This Model 1610A list display shows address and data flow on an HP 21MX S-bus.

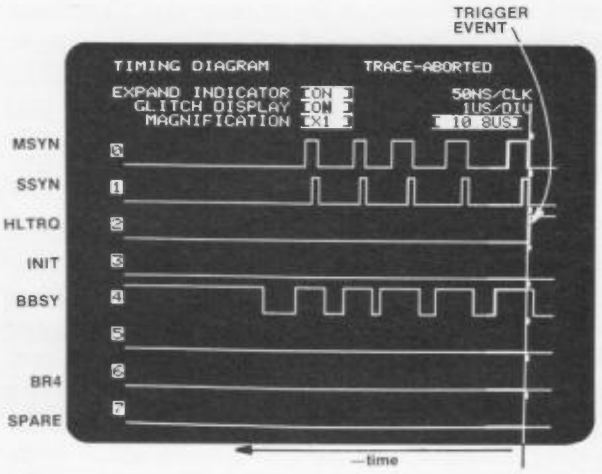


Figure 4. A Model 1615A timing display shows activity on control lines during a typical bus transaction.

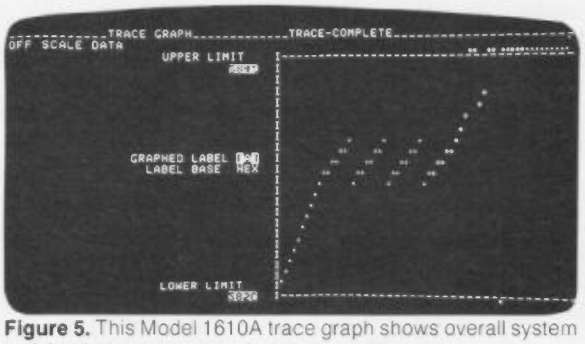


Figure 5. This Model 1610A trace graph shows overall system activity during a program loop.

FUNCTIONAL MEASUREMENTS

Common functional measurements include elapsed time of program execution for two systems (the benchmark technique), elapsed time of execution for two programs on the same machine, and an event count for a particular routine. These and similar measurements provide quantitative information as to comparative merit of instruments, efficiency of software code, likely sites of malfunction, etc.

INTERACTION WITH OTHER INSTRUMENTS

In design, implementation, and troubleshooting, it is frequently convenient to use an analog measurement instrument, e.g., an oscilloscope, in conjunction with the logic analyzer. All Hewlett-Packard logic analyzers provide some form of a trigger output based on a defined state or timing condition. A common technique is to pinpoint a faulty piece of hardware through an analysis of state flow, and then use the trigger capabilities of the logic analyzer to trigger an oscilloscope to study waveforms at that point. The trigger output is also useful for gating clocks, interrupting the system activity, triggering "clock stopper" circuits, or halting the system at breakpoints for static debugging.

MINICOMPUTER ARCHITECTURES

Basically, minicomputer architectures fall in two groups by manner of operation, asynchronous and synchronous. Asynchronous operation transmits signals under control of interlocked handshake signals. In synchronous operation, sequence of transmissions is controlled by equally-spaced clock signals.

ASYNCHRONOUS OPERATION

Typical of asynchronous systems is a DEC PDP-11/04® (figure 6), which uses a single 56 line UNIBUS® for communication between the CPU, memory, and any peripheral devices. Communication between devices is defined by the sequence of states on a set of interlocked UNIBUS control lines, the handshake sequences. A protocol of handshake sequences sets the master-slave relationship at any given time, and specifies what type of information is being passed and where. Another asynchronous system is the DEC LSI-11®. While the PDP-11 UNIBUS has separate address and data lines, the LSI-11 Q-BUS® multiplexes address and data on the same lines.

DEC PDP-11 Architecture

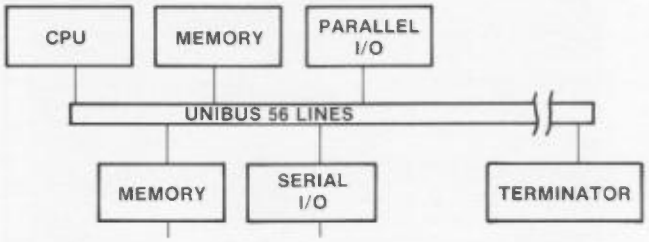


Figure 6. DEC PDP-11 architecture is a typical example of a system which operates asynchronously.

SYNCHRONOUS OPERATION

Systems which operate synchronously use a system timing generator. Usually there are two buses, a higher-speed memory bus and an I/O bus for slower peripheral devices. An example of a synchronous system, the HP 21MX, is diagramed in figure 7. Other synchronous systems are Data General's NOVA 3®[1] and Micro NOVA®[2] systems. As with asynchronous systems, what goes where is determined by the sequence of states on control lines, but the time of transmisson is set by defined cycles in time.

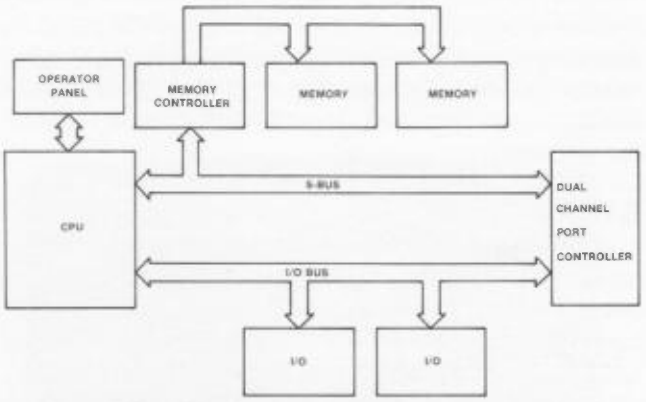


Figure 7. The HP 21MX block diagram illustrates one example of a system that operates synchronously.

SETTING UP THE LOGIC ANALYZER

The first step in viewing activity on a minicomputer with a logic analyzer is connecting the system to the analyzer. You always have the option of connecting 24 to 36 individual leads, but this procedure is time consuming and error prone. A more convenient way to connect the system and the analyzer is to use a dedicated interface. Hewlett-Packard now offers interfaces for some of the more popular minicomputers.

Interfaces available now include:

Model 10275A PDP-11 UNIBUS Interface
Model 10276A LSI-11 Q-Bus Interface
Model 10278A HP 1000 Series Interface
Model 52126A Intel MULTIBUS®[2] Interface

Interfaces will be soon available for Data General's NOVA 3 (HP Model 10279A) and MicroNOVA (HP Model 10280A). To further simplify the hookup, add Model 10277 General Purpose Probe Interface, as illustrated in figure 8. The probe interface includes interchangeable wire-wrap boards that allow the user to choose the address, data, and/or control lines to be monitored.

It is often desirable, and sometimes necessary, to pre-

Figure 8. The HP 10278A with an HP 10277 Probe Interface provides a method for easy, fast hookup between a logic analyzer and an HP Series 1000 minicomputer.

® Registered, Digital Equipment Corporation ®1 Registered, Data General ®2 Registered, Intel Corp.

process signals to the logic analyzer. The UNIBUS, for example, does not have a distinct clock signal associated with address or data lines, and operates asynchronously using handshake signals. A dedicated interface, Model 10275A PDP-11 UNIBUS Interface, derives a logic analyzer clock with decoding circuits on handshake signals. Some signal preprocessing is done directly on the interfaces. For a detailed description of the dedicated and general purpose interfaces, refer to the appendix.

STATE FLOW ANALYSIS

State flow measurements illustrate techniques for monitoring complex program execution in real time. For the examples that follow, the measurement system includes a Model 1610A Logic State Analyzer and the DEC PDP-11/04 Minicomputer connected with two interfaces, Model 10275A UNIBUS Interface and Model 10277A Opt 001 General Purpose Probe Interface.

BASIC STATE FLOW MEASUREMENT

The simplest measurement to make is to collect and display in-line code during program execution. The analyzer display is called a trace list. A line-by-line comparison of the 1610A trace list and the program assembler listing will reveal any inconsistencies between what is expected and what actually happens.

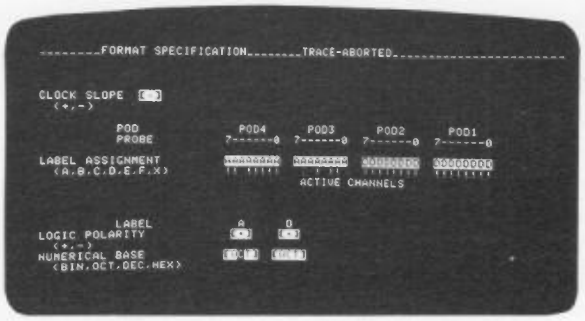


Figure 9. Model 1610A Format Specification selected for monitoring the DEC PDP-11/04 sets 16 channels for address and 16 channels for data, all in octal base.

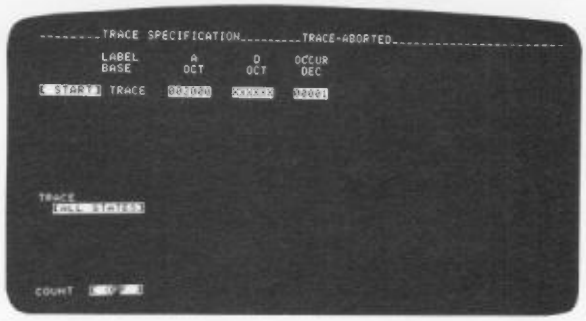


Figure 10. This is the Trace Specification for making a simple trace beginning at address 002000_8, and listing 64 sequential states.

```
                                WED, JUL 18, 1979, 10:30 AM

002000   010706                 BEGN MOV R7,P6       REFERENCE R6 TO THE PC
002002   062706  005776              ADD #n,P6       SET THE STACK POINTER (7776)
002006   010705                     MOV R7,P5       REFERENCE P5 TO THE PC
002010   062705  000470             ADD #n,P5       POINT TO 1ST ADD. IN INTR. MESS. (2500)
002014   010504                     MOV R5,R4       REFERENCE INTERRUPT MESSAGE 1ST ADDRESS
002016   012725  005015             MOV #n,(P5)+    MOVE LF,CR TO INTERRUPT BLOCK
002022   012725  051120             MOV #n,(R5)+    MOVE P,P TO INTERRUPT BLOCK
002026   012725  051505             MOV #n,(R5)+    MOVE S,E TO INTERRUPT BLOCK
002032   012725  020123             MOV #n,(R5)+    MOVE SB,S TO INTERRUPT BLOCK
002036   012725  047101             MOV #n,(P5)+    MOVE N,A TO INTERRUPT BLOCK
002042   012725  020131             MOV #n,(R5)+    MOVE SB,Y TO INTERRUPT BLOCK
002046   012725  042513             MOV #n,(R5)+    MOVE E,K TO INTERRUPT BLOCK
002052   012725  020131             MOV #n,(R5)+    MOVE SB,Y TO INTERRUPT BLOCK
002056   012725  047524             MOV #n,(R5)+    MOVE O,T TO INTERRUPT BLOCK
002062   012725  044440             MOV #n,(R5)+    MOVE I,SB TO INTERRUPT BLOCK
002066   012725  052116             MOV #n,(P5)+    MOVE T,N TO INTERRUPT BLOCK
002072   012725  051105             MOV #n,(R5)+    MOVE P,E TO INTERRUPT BLOCK
002076   012725  052522             MOV #n,(R5)+    MOVE U,R TO INTERRUPT BLOCK
002102   012725  052120             MOV #n,(P5)+    MOVE T,P TO INTERRUPT BLOCK
002106   012725  046440             MOV #n,(R5)+    MOVE M,SB TO INTERRUPT BLOCK
002112   012725  020105             MOV #n,(R5)+    MOVE SB,F TO INTERRUPT BLOCK
002116   005205                     INC R5          ADD ONE TO BLOCK
002120   010403                OVER MOV R4,P3       SET R3 = INTERPUPT MESSAGE ADDRESS
002122   112300                MESS MOVB(R3)+,P0    MOVE DATA BYTE TO XFER REG.
002124   004267  002650             JSR A    OUTT   OUTPUT DATA BYTE TO TERMINAL (5000)
002130   020305                     CMP R3,R5       ALL BYTES.OUT?
002132   001373                     BNE      MESS   NO,GET NEXT BYTE
002134   000005                     REST            RESET THE UNIBUS
002136   012737  002600  000060     MOV#n,@#A       LOAD INTR TRAP CELL WITH ADDRESS
002144   012737  000340  000062     MOV#n,@#A       LOAD NEXT CELL WI PRIORITY
002152   012737  000140  177776     MOV#n,@#A       SET PROCESSOR PRIORITY
002160   012737  000100  177560     MOV#n,@#A       TURN INTERRUPT ON
002166   000001                     WAIT            WAIT FOR INTERRUPT
002170   000167  177724             JMP A    OVER   START OVER (2120)
```

Figure 11. The program listing of a start-up routine can be compared to a trace list or graph of actual program execution on a logic analyzer.

The appropriate variables are set in the Format and Trace Specifications menus (figures 9 and 10) to produce a simple trace list beginning at the beginning of the start-up routine of figure 11. The resultant trace list, shown in figure 12, begins with the initial address, 2000_8, and contains the next 63 states in program execution. Twenty lines of state flow are shown on the analyzer at a time; the ROLL keys are used to view the other states of the trace list that are contained in the logic analyzer memory. A comparison of the trace list and the program listing confirm that program is executing properly.

GRAPH OVERVIEW

Another mode of viewing program execution on the 1610A logic analyzer is the graph display. Figure 13 is a graphic display of the first 64 addresses of the startup routine (figure 11). Each point on the graph corresponds to one of the 64 addresses (label A). Notice that the graph limits have been set at 2000 and 3000, the range of addresses in this routine, which facilitates viewing and interpretation. The points corresponding to the 20 states shown in the trace list mode are intensified on the graph display.

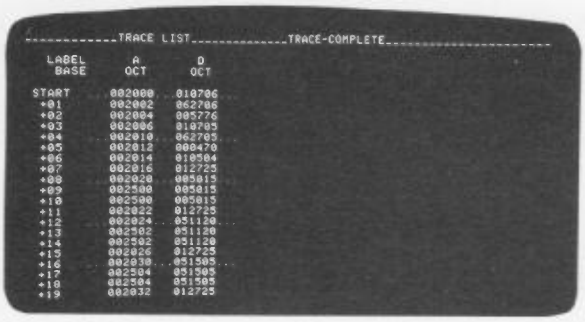


Figure 12. Using the trace specifications shown in figure 10 for the start-up routine of figure 11 results in a simple, in-line trace list of program execution.

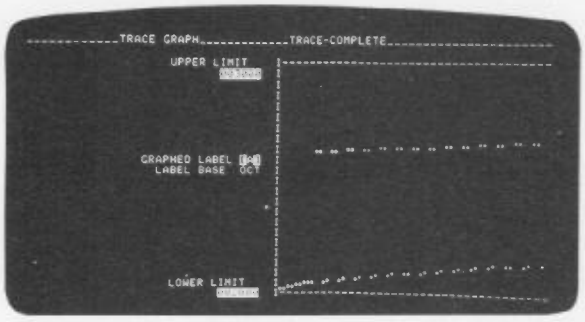


Figure 13. The 1610A graph display of the first 64 addresses (label A) of the start-up routine of figure 11 is bounded by 2000 and 3000, the range of addresses for the routine. Reading from the left, the first 20 points correspond to the 20 states displayed in figure 12, and these points are intensified.

In actual practice it's not always possible to specify in advance the particular 64 consecutive states that contain a subtle fault that makes a program crash or "go into the weeds". A larger overview is needed. This can be done by viewing a sample of states, every second state, every 17th state, or every nth state. For example, figure 14 is a graph of every fifth state of the same program, giving you an overview of activity across 320 states (5 x 64). The upper limit has been changed to 5100, thus showing the jump to an output routine at address 5000. By changing the occurrence count on the trace specifications you can "compress" the graph by any ratio from 1:1 to 65 536:1 for a graph of program activity in 64 states to **over four million states.**

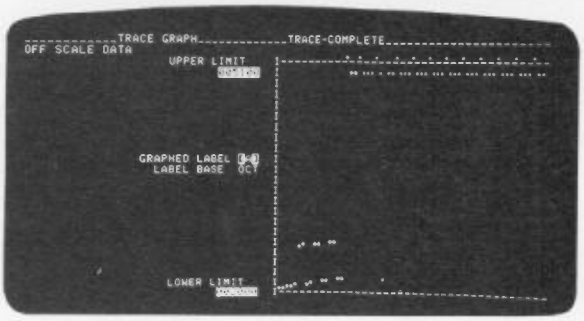


Figure 14. The expanded graph shows a discontinuity as the program jumps to the output routine at address 5000. Every 5th state of 320 states is shown as a point of the graph.

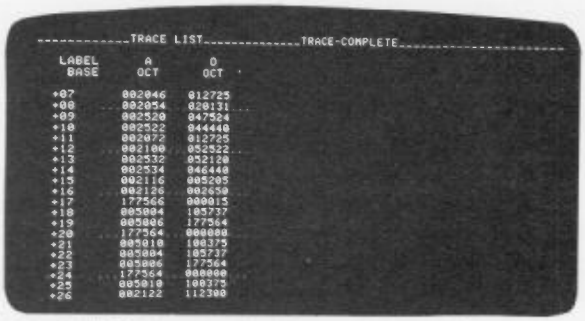


Figure 15. The trace list of addresses and data shows that the discontinuity of figure 14 occurs between lines 16 and 17.

MEASUREMENTS FROM GRAPH MODE

Recall that the 20 intensified points correspond to the 20 states shown in the trace list. Then, by moving the set of intensified points on the display to center around the apparent discontinuity in the graph of figure 14, and switching to the trace list (figure 15), it is clear that jump takes place after address 02126_8, line 16. Since the graph was every fifth state, the trace specification is changed to trace all states, centering the trace around the "suspect" address, and resulting trace list of figure 16 show a detailed state-by-state list around the jump. A few simple steps can locate a gross error and quickly narrow the field of investigation to a detailed list of states around the problem, a powerful set of techniques for rapid troubleshooting.

MEASUREMENTS ON SELECTED TYPES OF INFORMATION

Frequently, analysis is greatly simplified if only a particular type of information is collected. You may have indications that one I/O device is malfunctioning, or a

TRACE LIST ---------- TRACE-COMPLETE

LABEL BASE	A OCT	D OCT
-09	002112	012725
-08	002114	020105
-07	002536	020105
-06	002536	020105
-05	002116	005205
-04	002120	010403
-03	002122	112300
-02	002500	005015
-01	002124	004267
CENTER	002126	002650
+01	007776	165054
+02	005000	110037
+03	005002	177566
+04	177566	000000
+05	177566	000015
+06	005004	105737
+07	005006	177564
+08	177564	000000
+09	005010	100375
+10	005004	105737

Figure 16. Triggering on the point of discontinuity, 02126_8, and tracing all states centered on that address, gives a detailed view of the sequence of states around a previously identified problem area. From the program listing of figure 11, address 007776_8 is the stack pointer, and 005000_8 addresses are output addresses.

particular subroutine is suspect. Tracing only one kind of information is achieved with display qualification. With display qualification, you select for viewing only those transactions that are pertinent, which streamlines your troubleshooting and reduces the size of memory required for the logic analyzer.

One measurement dependent on display qualification is verfication that data is correctly written to the transmitter buffer on the serial I/O board. Continuing with the example of the start-up routine (figure 11), compare the message transferred beginning at address 2500_8 with the display on the terminal as a check on the operation of the serial I/O system. Changing the trace specifications to trace only states with addresses of $0025XX_8$ and setting the 10275A Interface switch to WRITE produces the trace list of figure 17 which is totally comprised of "write" instructions to the transmitter buffer. Comparing the trace list to the actual terminal display provides a check on data transmission to that buffer.

As another illustration of display qualification, suppose you wished to check only the output routine execution. In this case, set the Trace specification menu to trace ONLY STATES 177566_8, the address of the Transmitter Buffer Register on the serial I/O board, and set the qualifier switch of the 10275A Interface to WRITE. The printout of the contents of the logic analyzer

TRACE LIST ---------- TRACE-IN PROCESS

WARNING-SLOW CLOCK

LABEL BASE	A OCT	D OCT
START	177566	000015
+01	002500	005015
+02	002502	051120
+03	002504	051505
+04	002506	020123
+05	002510	047101
+06	002512	020131
+07	002514	042513
+08	002516	020131
+09	002520	047524
+10	002522	044440
+11	002524	052116
+12	002526	051105
+13	002530	052522
+14	002532	052120
+15	002534	046440
+16	002536	020105
+17		
+18		
+19		

Figure 17. The trace list contains only "write" instructions, and, when compared to the display on the terminal, verifies that the message has been correctly transferred to address 2500.

memory (with the addition of the translation of the data bits to alphanumeric characters) and the terminal display are shown in figure 18. These two examples are relatively simple applications of the process of display qualification.

TRACE LIST ---------- PRINT-IN PROCESS

LABEL BASE	A OCT	D OCT	
START	177566	000114	L
+01	177566	000040	SP
+02	177566	000062	2
+03	177566	000060	0
+04	177566	000060	0
+05	177566	000060	0
+06	177566	000015	CR
+07	177566	000015	CR
+08	177566	000012	LF
+09	177566	000044	$
+10	177566	000000	NUL
+11	177566	000123	S
+12	177566	000015	CR
+13	177566	000015	CR
+14	177566	000012	LF
+15	177566	000120	P
+16	177566	000122	R
+17	177566	000105	E
+18	177566	000123	S
+19	177566	000123	S
+20	177566	000040	SP
+21	177566	000101	A
+22	177566	000116	N
+23	177566	000131	Y
+24	177566	000040	SP
+25	177566	000113	K
+26	177566	000105	E
+27	177566	000131	Y
+28	177566	000040	SP
+29	177566	000124	T
+30	177566	000117	O
+31	177566	000040	SP
+32	177566	000111	I
+33	177566	000116	N
+34	177566	000124	T
+35	177566	000105	E
+36	177566	000122	R
+37	177566	000122	R
+38	177566	000125	U
+39	177566	000120	P
+40	177566	000124	T
+41	177566	000040	SP
+42	177566	000115	M
+43	177566	000105	E
+44	177566	000040	SP
+45	177566	177640	
+46			
+47			
+48			

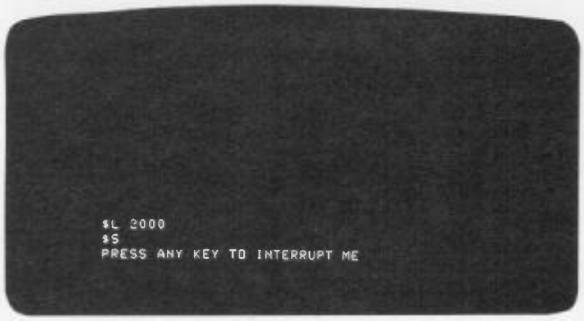


Figure 18. A column has been added to this 1610A print-out of a trace list (a) of writes to address 177566_8 to translate the data column to alphanumeric code. A comparison of the trace list and the display on the terminal (b) verify that the output routine is executing properly.

COMPLEX STATE MEASUREMENTS USING SEQUENTIAL TRIGGERS

Model 1610A Logic Analyzer permits the entry of up to seven words which must be found in sequence before a trace is begun. Each of the trigger words may be further qualified by specifying the number of times that word must appear (up to 65 536 times) before a search is made for the next sequence term, or the trace is begun. A sequence restart state may be used to restart the search for the given sequence of trigger words if the sequence does not occur before the sequence restart word condition. The two common situations which use these triggering capacities are tracing multipath code or tracing nested loops.

Tracing Multipath Code. It's a rare minicomputer program that doesn't include at least one branching network. In the sample program, the initial address of the routine that outputs an interrupt message, 2120_8, can be reached by three paths (figure 19). Suppose you are only interested in this routine when it occurs following address 2714_8. With sequential triggers, this can be done simply by listing addresses 2714_8 and 2120_8 as the two terms to be found in sequence. The Trace Specification menu for this condition is shown in figure 20, and simply lists the two addresses, each occurring once. The trace list will show the two sequence terms and list the subsequent states. Should you wish to view the states occurring between the two sequence terms, change the Trace Specification menu from START trace mode to CENTER trace mode. In the trace list shown in figure 21, only four states occurred between the two terms. Obviously, with the option of up to seven trigger words, it is easy to specify only the branch of interest in almost any situation.

Tracing Nested Loops. The capability for sequential triggers is also a convenient feature when analyzing information which is contained in nested loops. One routine in the sample program repetitively writes a space and all 94 printable ASCII characters to the terminal (figure 22). Each time the major loop, J, occurs, the minor loop, K, is executed 94 times. For an exercise, consider how best to trace program execution following the 36th occurrence of the K loop during the fifth pass of J loop. Most logic analyzers have a trigger delay, i.e., the number of states to be skipped following the occurrence of a trigger word. Then, using a little arithmetic, you can trace the desired portion of program execution by using a trigger address of 20062_8, and a trigger delay of 412 states (4 x 94 + 36) **assuming the loops are of fixed length.** A better way is using sequential triggers as shown in the Trace Specification menu of figure 23. The 1610 Logic Analyzer begins at address 20000_8 outside the loops, passes the initial address of the major loop five times, and then traces program execution after the address of the minor loop is passed 36 times. Note that in this case, occurrences of the address of the minor loop are counted. The resulting trace is shown in figure 24; the count of occurrences of address 20062_8 is shown in the right-hand column. The first count, 376 (4 x 94), shows that four complete passes of the major loop occurred before the 36 passes of the minor loop were counted. Sequential triggers make it simple to trace information deep in nested loops of fixed or **variable lengths** without the bother and potential errors of using trigger delay and "simple" arithmetic.

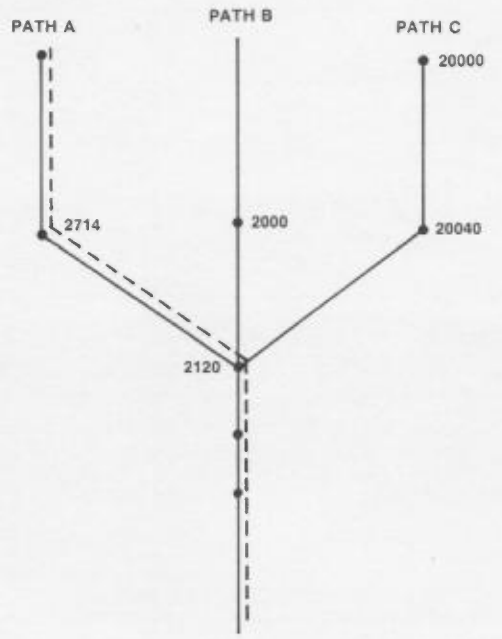


Figure 19. The interrupt message routine at address 2120_8 can be entered by any one of three different paths.

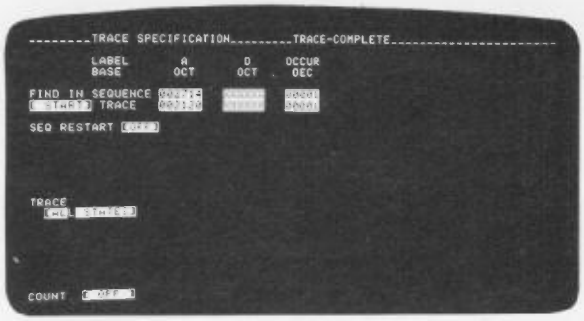


Figure 20. Trace specification for capturing the interrupt message output routine following entry from path A, (figure 19) which includes an address, 2714_8, which is unique to path A.

Figure 21. Changing the Trace Specification menu of figure 20 to CENTER trace mode produces a trace list which shows that four states occurred between the two sequential trigger words, addresses 2714_8 and 2120_8.

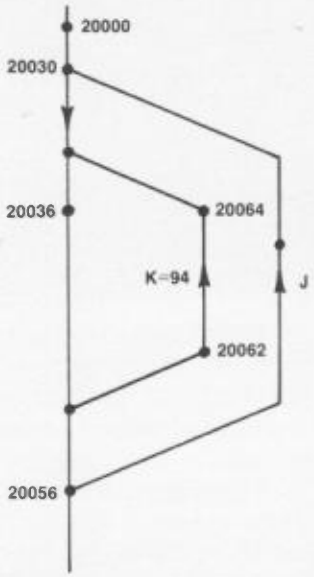


Figure 22. A nested loop contains a minor loop K that is executed 94 times in each pass of the major loop J.

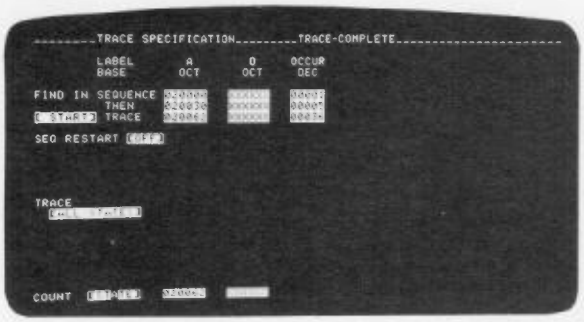


Figure 23. With this Trace Specification, the 1610A captures program activity in the nested loop of figure 22.

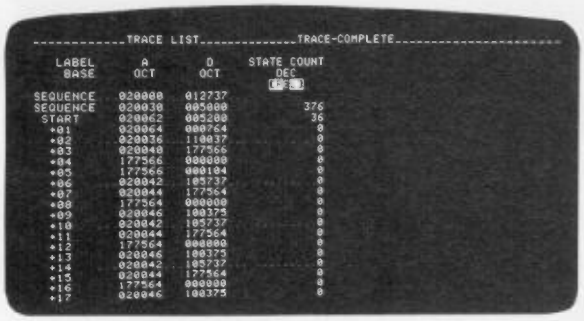


Figure 24. The Trace Specification menu of figure 23 produces a trace list of a nested loop.

MEASURING EXECUTION TIME

The count time measure is particularly useful for benchmark tests to check for program efficiency. As an example, the I/O service routine shown in figure 25 can be evaluated using time measures available on Model 1610A logic state analyzer. This routine moves a 12-character string to a buffer, and then moves the characters one by one to a terminal, with 150 ms between each transmission. The Trace Specifiation menu (figure 26) is set to count TIME, and the time count is set to relative (REL) on the associated trace list (figure 27). Relative time counts are listed for each state, and each time is measured between the state and the preceding listed state. Figure 28 shows the same trace list with an absolute (ABS) time measure, and all times are measured from zero at the trace point.

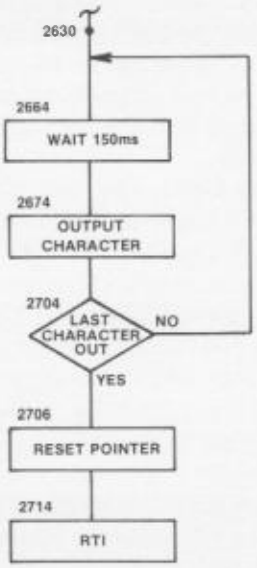


Figure 25. An I/O service routine can be evaluated for efficiency and correct execution using time measures available on Model 1610A logic analyzer.

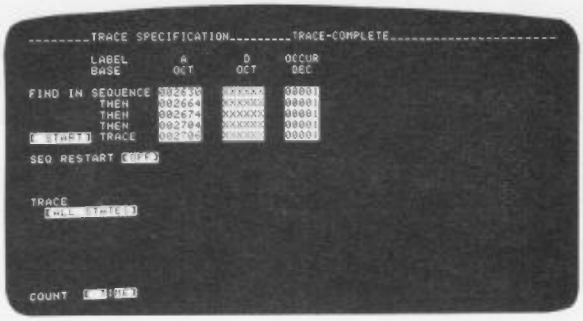


Figure 26. The Trace Specification menu sets the 1610A to measure a delay loop and the time required to transmit a character for the routine of figure 25.

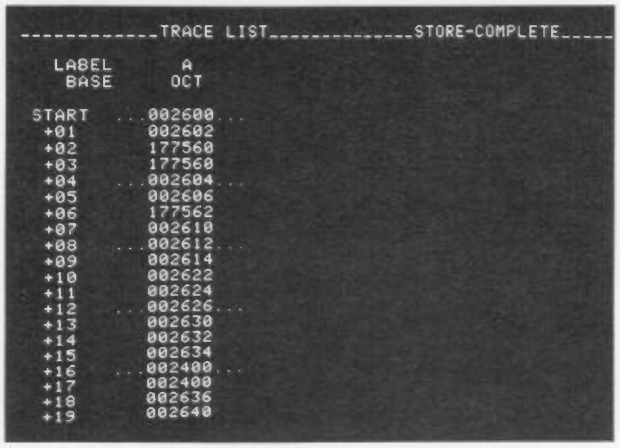
Figure 27. The trace list showing the relative measured times of execution for the routine of figure 25 shows the elapsed time between sequential states on the display.

```
----------TRACE LIST----------TRACE-COMPLETE----------

  LABEL      A        D        TIME
   BASE     OCT      OCT       DEC
                              CH-D

SEQUENCE  002630   010503   -  1.760 S
SEQUENCE  002664   012701   -  1.760 S
SEQUENCE  002674   112300   -  1.614 S
SEQUENCE  002704   001367   -  1.614 S
 START    002706   005005      0    US
  +01     002710   062705   +  2.5    US
  +02     002712   002542   +  3.9    US
  +03     002714   000002   +  6.7    US
  +04     007764   002170   +  9.2    US
  +05     007766   000140   +  9.8    US
  +06     002170   000167   + 11.9    US
  +07     002172   177724   + 13.3    US
  +08     002120   010402   + 15.6    US
  +09     002122   112300   + 18.4    US
  +10     002500   005015   + 19.9    US
  +11     002124   004267   + 22.4    US
  +12     002126   002650   + 23.8    US
  +13     007766   165054   + 25.7    US
  +14     005000   110037   + 28.5    US
  +15     005002   177566   + 30.4    US
```

Figure 28. The absolute time display shows the accumulated transmit time for the complete trace, using the final trigger word as time zero.

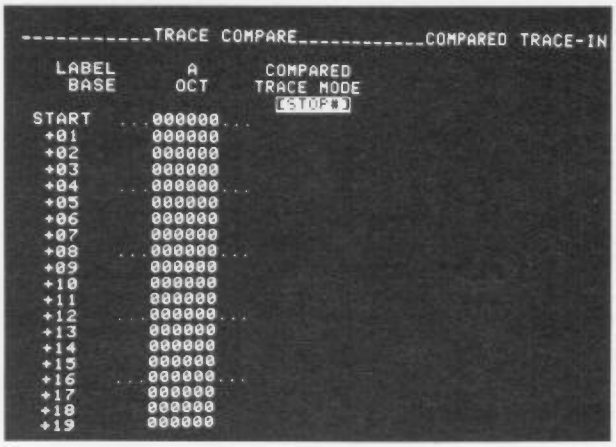
LOCATING INTERMITTENT ERRORS

Intermittent errors are some of the most difficult problems to pinpoint. Results from the error could show up far downstream from the error, and the original error might be missed. Model 1610A has a Trace Compare mode which can be applied in situations where an error condition appears sporadically. First, the likely location of the error is defined, and a trace is made of a code segment which is known to be executing properly (figure 29a). This correct trace is stored in the logic analyzer. The three choices for subsequent traces, shown in inverse video, are OFF, STOP =, or STOP #. Using the "stop if not equal" (STOP#) operation, so long as a new trace and the stored trace are the same, the display show only zeros (figure 29b). Suppose the new trace does not match the stored trace (figure 29c); then the analyzer halts and the display will contain nonzero numbers (figure 29d). These numbers, when converted to binary numbers, specify which binary bits are nonmatching. For example, in figure 29d, line 10 is 34_8. Converted to binary, $000034_8 = 0\ 000\ 000\ 000\ 000\ 011\ 100$, and bits 2, 3, and 4 of line 10 of the new trace do not match the corresponding bits of line 10 of the stored trace list.



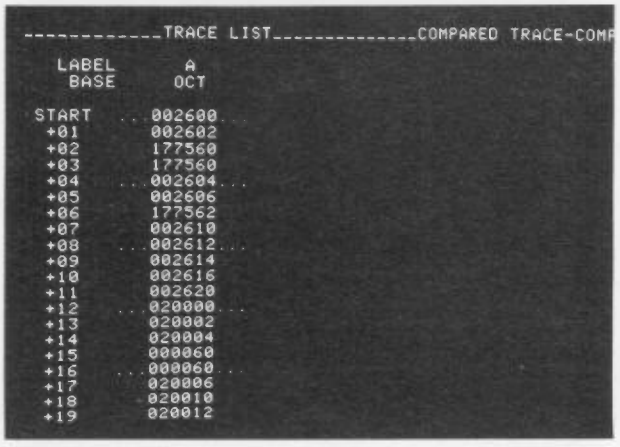
```
----------TRACE LIST----------STORE-COMPLETE-----

  LABEL      A
   BASE     OCT

 START   002600
  +01     002602
  +02     177560
  +03     177560
  +04     002604
  +05     002606
  +06     177562
  +07     002610
  +08     002612
  +09     002614
  +10     002622
  +11     002624
  +12     002626
  +13     002630
  +14     002632
  +15     002634
  +16     002400
  +17     002400
  +18     002636
  +19     002640
```

a



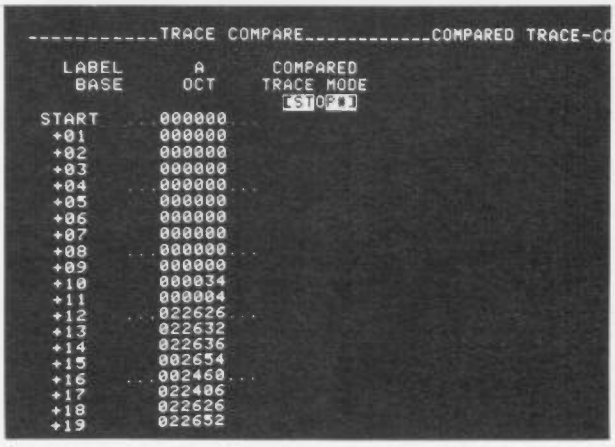
```
----------TRACE COMPARE----------COMPARED TRACE-IN

  LABEL      A       COMPARED
   BASE     OCT     TRACE MODE
                    (STOP#)

 START   000000
  +01     000000
  +02     000000
  +03     000000
  +04     000000
  +05     000000
  +06     000000
  +07     000000
  +08     000000
  +09     000000
  +10     000000
  +11     000000
  +12     000000
  +13     000000
  +14     000000
  +15     000000
  +16     000000
  +17     000000
  +18     000000
  +19     000000
```

b



```
----------TRACE LIST----------COMPARED TRACE-COMP

  LABEL      A
   BASE     OCT

 START   002600
  +01     002602
  +02     177560
  +03     177560
  +04     002604
  +05     002606
  +06     177562
  +07     002610
  +08     002612
  +09     002614
  +10     002616
  +11     002620
  +12     020000
  +13     020002
  +14     020004
  +15     000060
  +16     000060
  +17     020006
  +18     020010
  +19     020012
```

c



```
----------TRACE COMPARE----------COMPARED TRACE-CO

  LABEL      A       COMPARED
   BASE     OCT     TRACE MODE
                    (STOP#)

 START   000000
  +01     000000
  +02     000000
  +03     000000
  +04     000000
  +05     000000
  +06     000000
  +07     000000
  +08     000000
  +09     000000
  +10     000034
  +11     000004
  +12     022626
  +13     022632
  +14     022636
  +15     002654
  +16     002460
  +17     022406
  +18     022626
  +19     022652
```

d

Figure 29. In Compare Trace mode, using STOP #, a known "good" trace (a) can be repetitively compared with the current trace. When a subsequent trace matches the stored trace list, the display will show only 0's (b); but, if a new trace list does not match the stored trace list (c), the Trace Compare display (d) will contain nonzero numbers for those states not matching.

TIMING ANALYSIS

Most problems in a digital system can be located with techniques that monitor the state flow. However, there is a subset of problems that are best resolved through timing analysis. Timing is often the most crucial factor in difficulties related to handshake sequences, control lines, glitches, clock phasing, and similar problems. To illustrate the use of a logic analyzer in timing analysis, three examples of common measurements are given. For measurements on control lines and glitch detection, Model 1615A Logic Analyzer is used.

ANALYSIS OF TIMING RELATIONSHIPS OF CONTROL LINES

A system crash always demands immediate attention and resolution. In this example, each time a program is started at address 60 000, the PDP-11/04 halts and the run light goes out. Analysis of both address and control lines is required.

The first step is to make the appropriate connections between the minicomputer and the logic analyzer. The lower 16 bits of address are assigned to Pods 3 and 2, and, with the wire-wrap board of the 10277B interface, control lines are attached to Pod 1 as follows:

Bit 0 - MSYN
Bit 1 - SSYN
Bit 2 - HLTRQ
Bit 3 - INIT
Bit 4 - BBSY
Bit 5 - NPR
Bit 6 - BR4
Bit 7 - Spare

Like Model 1610A, Model 1615A logic analyzer is set up with menus. For this problem, the Format Specification menu of figure 30 and the Trace Specification menu of figure 31 are used. If the program were executing properly, the sequence of steps would include:

1. 060000 is asserted on the address bus
2. MSYN is set to 1 by the CPU
3. Correct data is set to the data bus
4. SSYN is set to 1 by ROM
5. Data is read by CPU
6. MSYN is set to 0 by CPU
7. SSYN is set to 0 by ROM

No trace list is generated, so it is apparent that the trigger condition address 060000, is never met. While still in this post "crash" state, it could be worthwhile to look at the state of the control lines. Changing to timing mode and triggering on any event under label E result in the display of figure 32 showing line 2, HLTRQ (halt request) line, is high, which of course, halts the system.

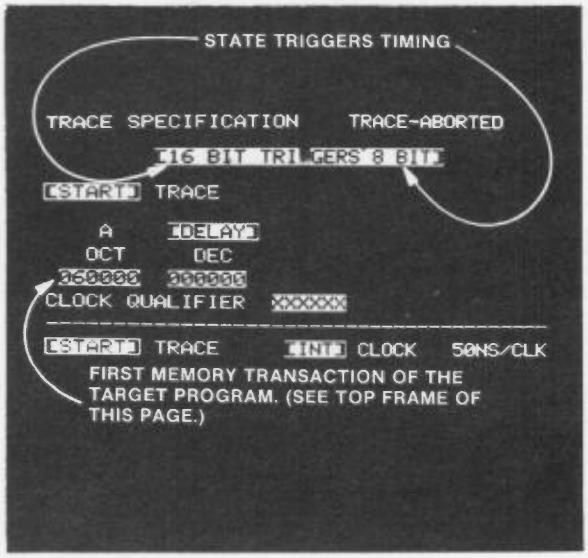


Figure 31. This trace specification menu will initiate a trace list on the 1615A at first address in the target program, 060000_8.

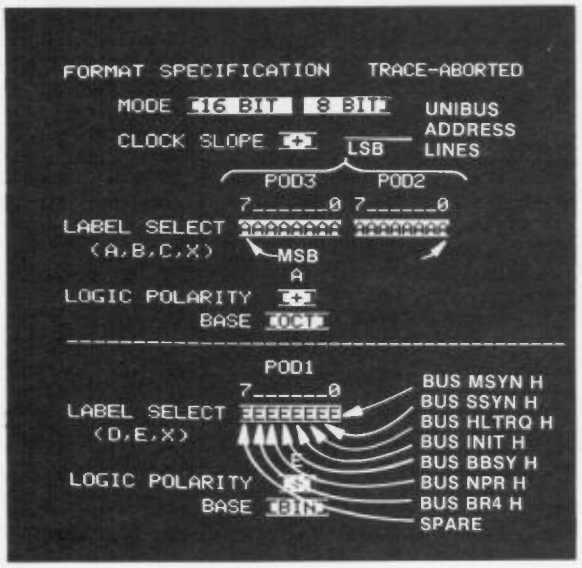


Figure 30. This format specification allows display of both state and timing information on Model 1615A Logic Analzyer.

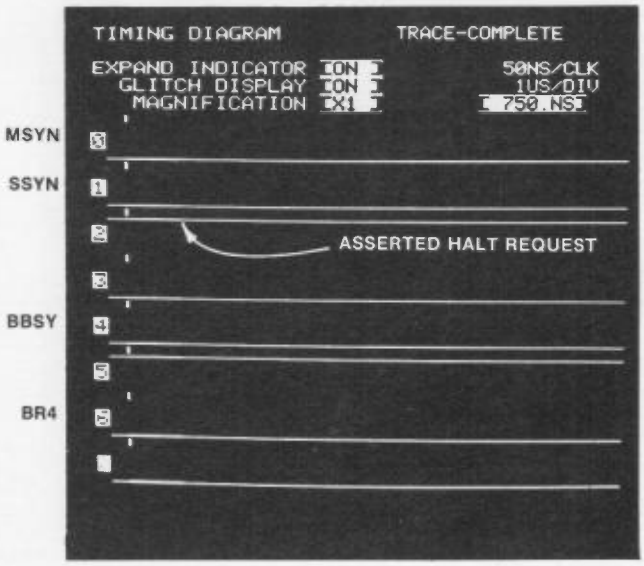


Figure 32. The timing diagram of control lines after the system crash shows line 2, HLTRQ (halt request), is high, and a likely result of the crash.

The next puzzle is why the HLTRQ line went high. Choosing END trace mode and a trigger on HLTRQ will show what events took place before the line went high. When the test is run again, the timing display of figure 33 shows the assertion of line 2, HLTRQ, at the far right of the display. Notice lines 0 (MSYN) and 1 (SSYN); line MSYN was asserted by the CPU, but a response on line SSYN was not returned by the memory. Protocol for PDP-11/04 control lines declares a time-out error if a MSYN signal is placed on the UNIBUS and an SSYN response is not made within 20 μs. The time-out error causes the current instruction to be aborted and the processor "traps out". Address 60000 is on a special ROM memory board which has a hold-off circuit to compensate for the access time of the ROM. A likely problem source could be that the one-shot RC time constant is too long. Connecting the spare line input (line 7) to the one-shot output and repeating the test gives the display in figure 34. Then the time the one-shot output was asserted is measured, and the direct time read-out is 21.8 μs; this exceeds the time-out limits for the UNIBUS. Reducing the RC time constant in the hold-off circuit removes the problem, and four steps were all that were required to identify the problem.

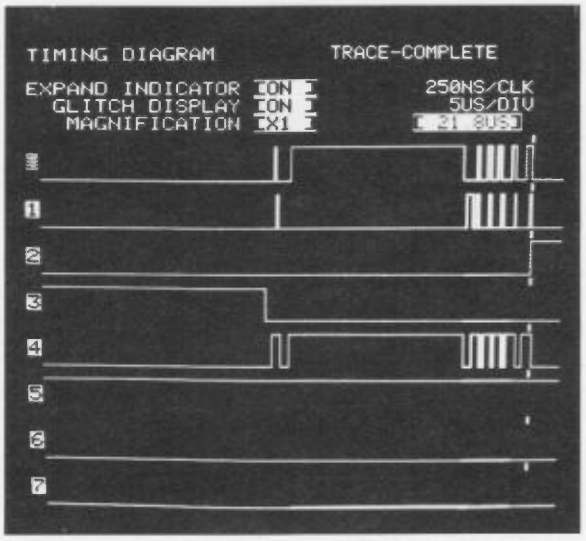


Figure 33. When data preceding the crash is collected it is seen that MSYN (line 0) was asserted, but SSYN (line 1) was not returned by the memory.

GLITCH DETECTION

A glitch is a transient signal, and if it has sufficient amplitude and duration, it can cause a problem in a system. Using the program and setup of the last example, an analysis will be made of a different problem. In this illustration, a terminal service routine is interrupt-driven and is entered by pressing any key on the terminal. The problem is that each time a key is pressed, the system halts. To troubleshoot this problem, the setup and procedures of the preceding example are followed, and, as before, a display of the quiescent states of the control lines following the crash show that the HLTRQ line has been asserted. This time, triggering on the HLTRQ line and viewing the preceding activity on the controls shows no abnormal activity on the control lines (figure 35).

A capability for both state and timing analysis is a particular advantage for troubleshooting problems of this nature. The sequence of events suggests that a glitch could be involved, but an error in the software should not be overlooked. Model 1615A Logic Analyzer defines a glitch as multiple transitions across threshold between sampling periods. If a glitch occurs, it appears as a vertical bar, brighter and wider than the timing lines, and can be easily distinguished even if it occurs at a timing transition.

For this analysis, the next step is to check activity on control and address lines before HLTRQ is asserted.

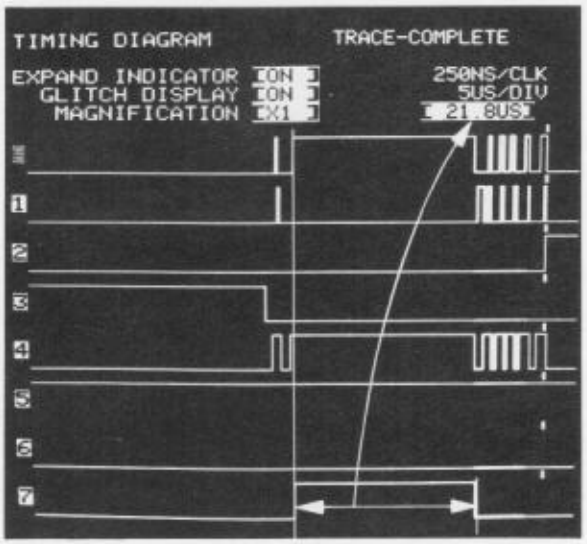


Figure 34.. After attaching the spare lead to the ROM hold-off output and measuring the time with the 1615A curser shows an elapsed time of 21.8 μs, the time line 7 was asserted.

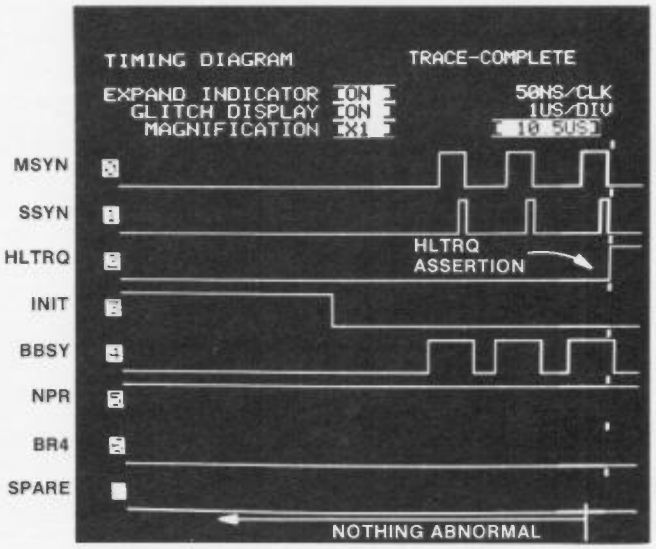


Figure 35. Trigger on the HLTRQ line, line 2, and displaying the activity preceding the assertion of HLTRQ shows no abnormal activity on the other control lines.

This is done with the Format Specification menu of figure 36 and the Trace Specification menu of figure 37, in which the timing measure triggers the state measure, and the activity preceding the trigger is to be displayed. The timing diagram shows no abnormal activity on control lines monitored, but the trace list (figure 38) shows the addresses of power-fail traps cells, 000024 and 000026, on lines 252 and 253. In normal operation, whenever the power drops below a specified level, either the AC LO or DC LO signals (generated by one-shots) are asserted and the CPU automatically traps to a power failure routine at location 000024. For this example, the spare line is connected to the AC LO line.

The Trace Specification menu is altered to set a time delay of 15 μs to display activity before and after the trigger on the assertion of HLTRQ. Rerunning the test and viewing the timing diagram shows (figure 39) that a power-fail signal does appear on the AC LO line. Moreover, this signal appears only after the interrupt from the terminal and coincides with the bus request for interrupt on the BR4 line (line 6). The two signals are of unequal duration, which suggests they are related, but not tied together. Two likely causes are (1) a wiring error across the path of the one-shot or (2) a glitch on the input to the AC LO one-shot. Since the latter problem is more likely, another line is assigned to the input of the one-shot. Line 5, NPR, is used because this channel has been inactive in the previous measurements. Then, the Trace Specification is set to trace a glitch on line 5, which is now the input of the one-shot. This timing diagram (figure 40) shows a glitch on the input to the power-fail circuitry which occurs at the same time as the interrupt signal from the terminal on line 6. A more detailed view (figure 41) is gained using the times ten

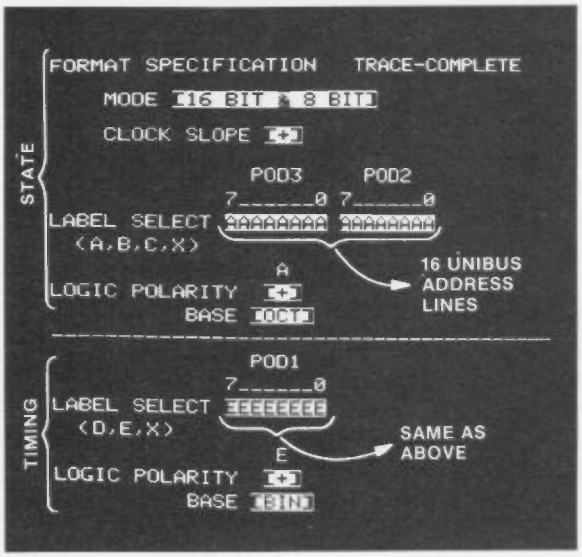


Figure 36. A Format Specification menu is set to trace both state and timing.

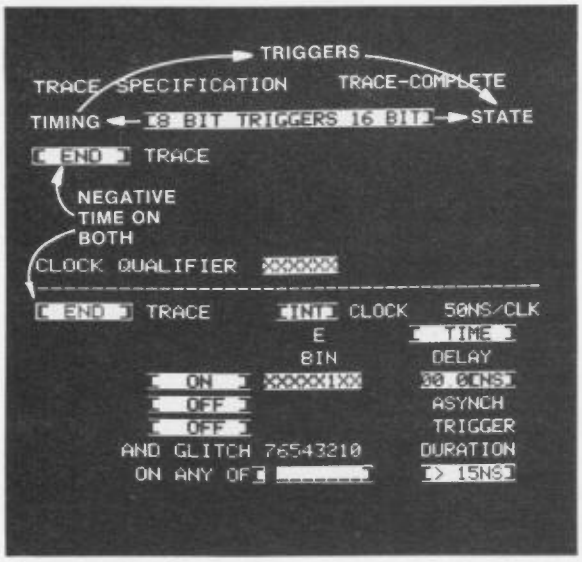


Figure 37. The Trace Specification menu sets the timing to trigger the state measurement, and putting the trigger at the END of the trace, collects and displays activity occurring before the HLTRQ line is asserted.

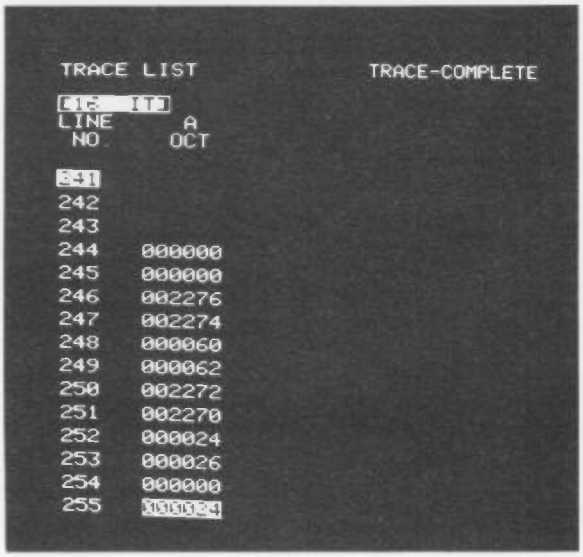


Figure 38. The trace list associated with the timing diagram of figure 37 shows the power fail trap cells on lines 252 and 253.

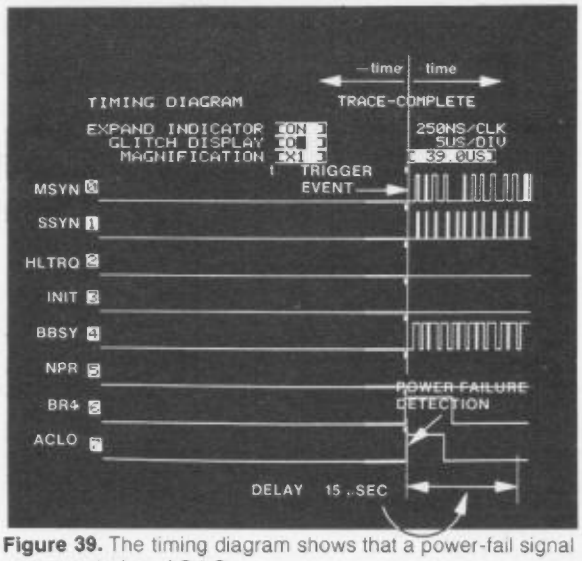


Figure 39. The timing diagram shows that a power-fail signal was asserted on AC LO.

(X10) magnifier, confirming the concurrence of the glitch and the interrupt signal. In this example a capacitive coupling of the two lines causes the glitch, and consequently, the system halt. The problem can be corrected on the PC board. The interactive use of time and state traces reduces a complex problem through a short series of logical steps, each more narrowly defining the range of likely problem sources.

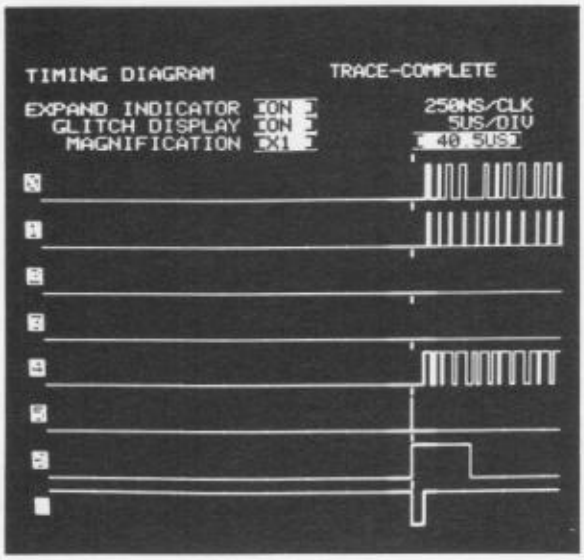


Figure 40. The timing diagram shows a glitch on line 5, which is now the trace of activity of the input to the one-shot.

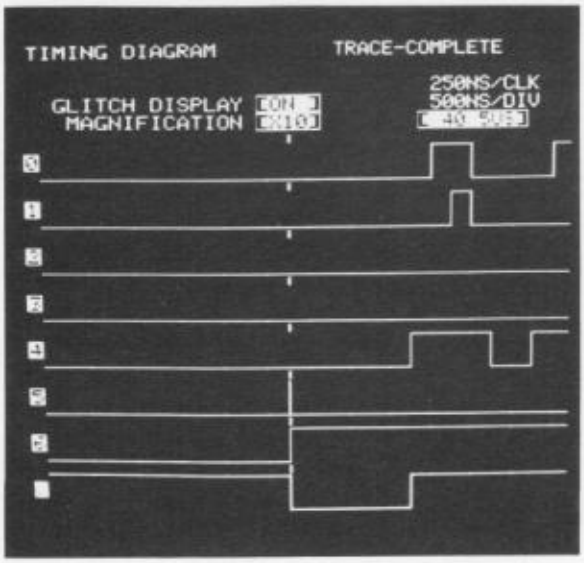


Figure 41. A magnification (X10) of the timing diagram of figure 40 confirms that the glitch on the AC LO line (line 5) coincides with the assertion of the interrupt signal from the terminal on the BR4 line (line 6).

ASYNCHRONOUS MEASUREMENTS WITH A LOGIC STATE ANALYZER

The first two examples of timing analysis used Model 1615A which performs both state and timing traces. The first problem, a time-out problem could also have been resolved using state flow with a Model 1610 Logic State Analyzer.

The 10 MHz oscillator output of the 1610 is used to clock data into the logic analyzer and the Format Specification of figure 42 is set. Variables to be traced include the address (in octal base) under label A, and seven control lines in binary base, including:

MYSN (label B)	HLTRQ (label E)
SSYN (label C)	ACLO (label E)
BR4 (label D)	BBSY (label F)
INTR (label D)	

The Trace Specification menu (figure 43) is set to follow the sequence that should occur beginning with address 60 000. The trace that results (figure 44) shows that address 60 000 is on the address bus, but when MSYN (label B) is asserted, the correct response on the SSYN line (label C) does not occur, and the third term of the sequence of triggers is not satisfied. Changing the Trace Specification, and placing an X (don't care) in the sequence where SSYN originally appeared, and using a

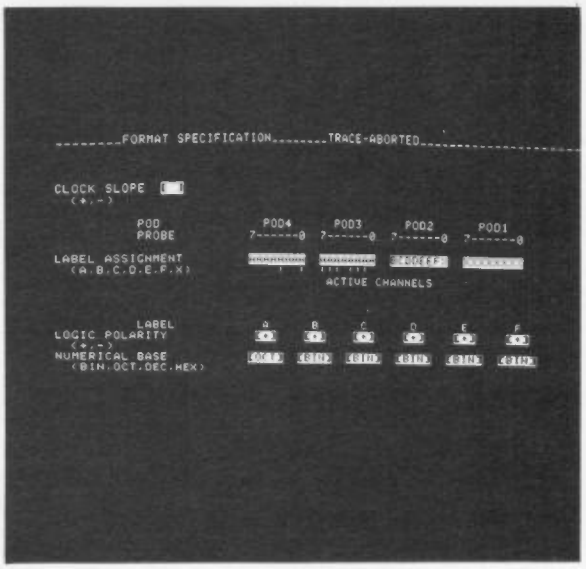


Figure 42. The Format Specification assigns channel labels for the address and control lines of interest. The address is assigned octal base, and the control lines are in binary base.

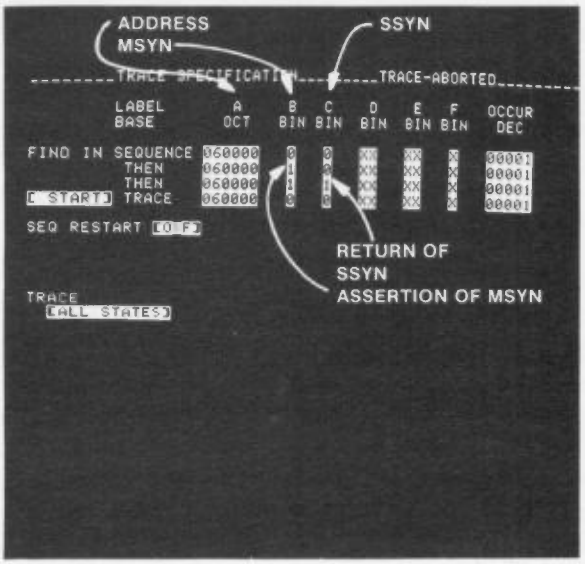


Figure 43. The Trace Specification rigidly follows the sequence of event that should occur at address 60000.

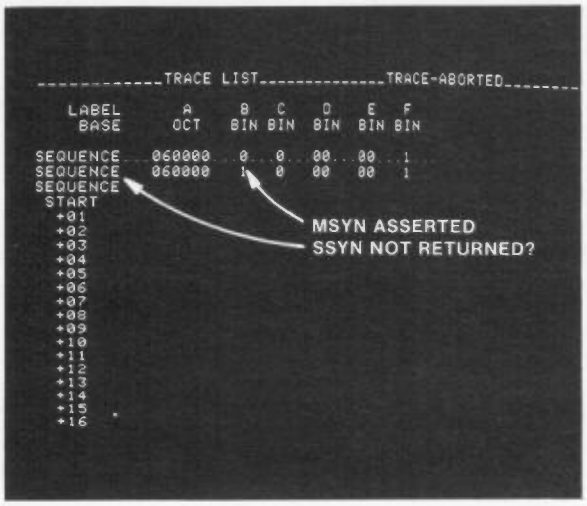


Figure 44. The trace list shows that SSYN was not returned in response to MSYN at address 60000.

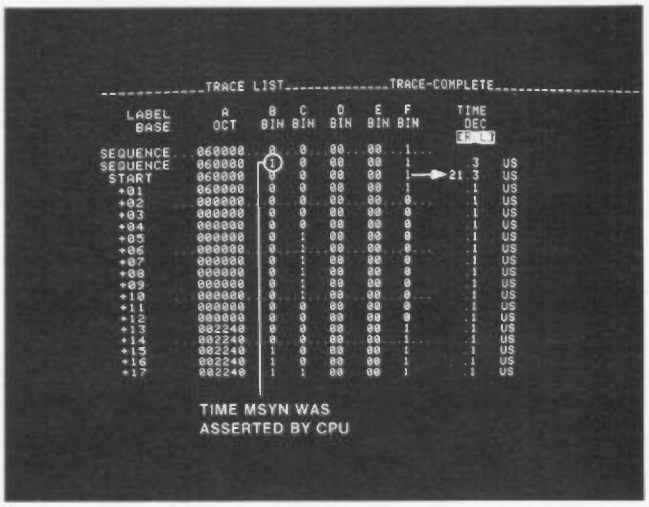


Figure 45. The trace list with a time measurement shows that MSYN is asserted for 21.3 μs.

TIME count, produces the trace of figure 45. On the third line of the trace, the last of the sequence terms, the relative time count is 21.3 μs. But this exceeds the 20 μs limit of the UNIBUS, and using the same logic as was used in the first example with Model 1615A Logic Analyzer again leads to the deduction that the one-shot RC time constant on the SSYN hold-off circuit is set too long. The time measures and internal clock make the Model 1610A Logic State Analyzer adaptable to timing analysis as well as state flow analysis.

SUMMARY

The real-time analysis capabilities of logic analyzers make them excellent tools for troubleshooting minicomputer systems. Analysis is even further simplified by adding the appropriate interfaces for easier connections and preprocessed signals. Logic analyzers can be used with synchronous and asynchronous minicomputer architectures. Examples show the advantages of adding logic analyzers to your set of troubleshooting tools for both state and timing analysis in minicomputer systems.

APPENDIX: MINICOMPUTER INTERFACE DESIGN CONSIDERATIONS

The design of a minicomputer interface begins with a fairly good knowledge of the particular bus to be analyzed. In the typical minicomputer there are usually 16 bits of address, 16 bits of data, and up to 30 or more control lines with data transfers occurring at rates varying between 100 kHz and 10 MHz. This plethora of information must somehow be interfaced to the input requirements of the logic analyzer and in such a fashion as to be easily understood by the user. The design must take into consideration such things as clock generation, qualifiers, data timing, bus loading, and general logic analyzer requirements.

Generation of a clock to the analyzer usually requires ORing several control lines together for capturing all transactions on the bus. Figure A-1 shows the simple case of two separate control lines being ORed together. The delay line provides for the minimum clock pulse width requirement of the logic analyzer to be satisfied. A more typical example of clock generation is shown in figure A-2 where the data-in pulse must be delayed in order to satisfy the data setup time of the logic analyzer.

Since logic analyzers are typically up to 36 bits in width, it is useful to generate transaction qualifiers,

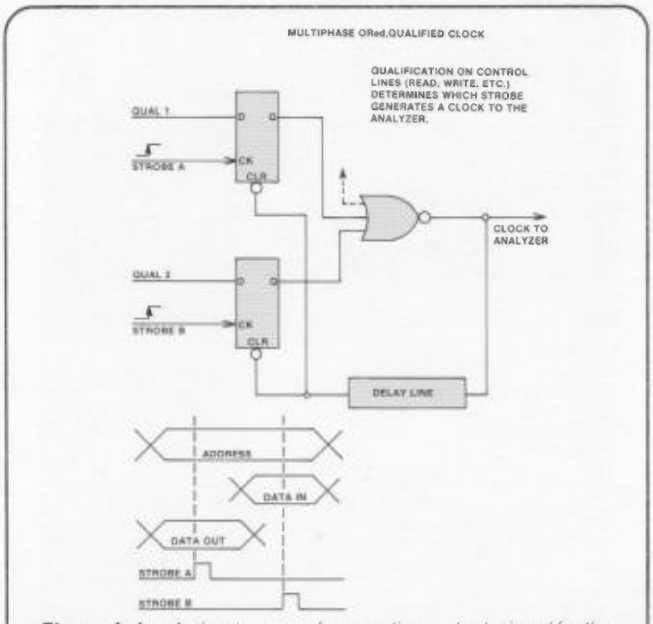


Figure A-1. A simple case of generating a clock signal for the logic analyzer ORs two control lines and adds a sufficient delay to meet the minimum clock pulse width requirement of the analyzer.

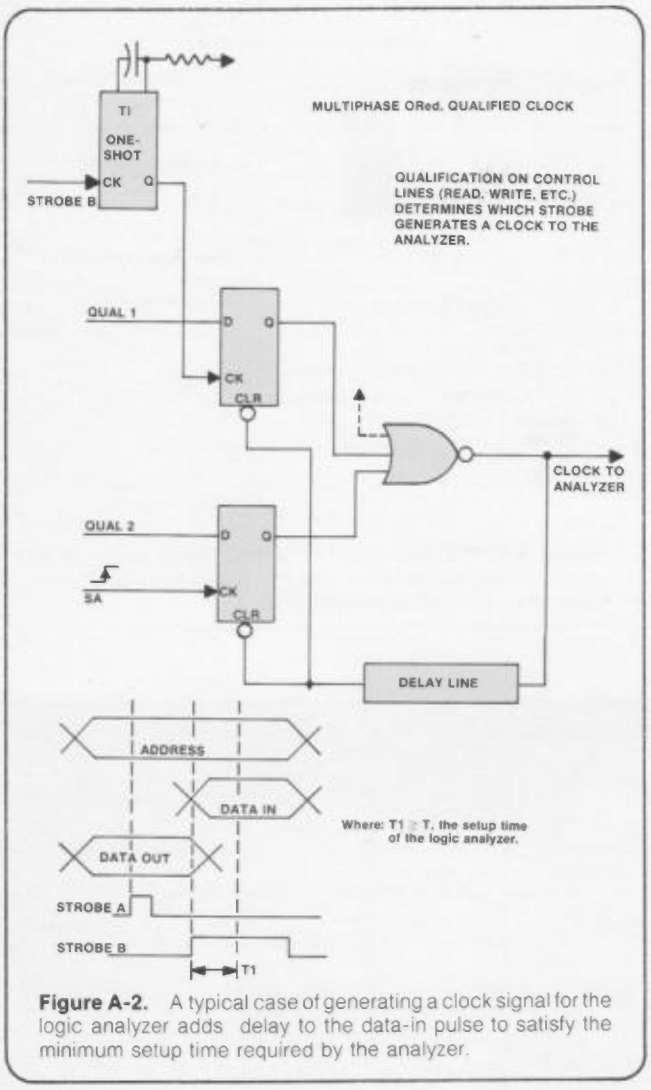


Figure A-2. A typical case of generating a clock signal for the logic analyzer adds delay to the data-in pulse to satisfy the minimum setup time required by the analyzer.

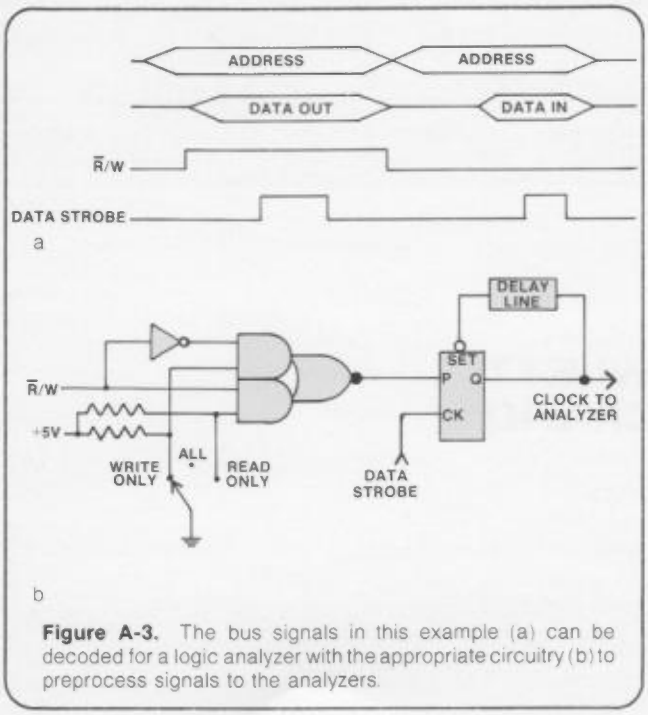


Figure A-3. The bus signals in this example (a) can be decoded for a logic analyzer with the appropriate circuitry (b) to preprocess signals to the analyzers.

such as read/write, DMA, memory refresh, etc. This requires circuitry to decode the transaction type and then to optionally inhibit the associated clock to the analyzer on the basis of some switch setting on the interface. In this manner, only transactions of a certain type will be captured or excluded for analysis without the need for additional channels for qualification. This data reduction decreases the number of states to be analyzed to locate the desired state. Figure A-3 shows some sample bus signals and circuitry for decoding the transactions.

Data on the bus can be transferred in either a synchronous or an asyncronous manner, depending upon the minicomputer.* In most synchronous machines there are two buses, one for memory and a slower bus for I/O. In this case, the two buses have to be multiplexed together (figure A-4) in order to maintain time relationships in the list display and to accommodate the logic analyzer data width. Either bus type might incorporate a multiplexed address/data bus. With a multiplexed bus, the address is first placed on the bus and then some time later the data is placed on the bus (figure A-5). This requires a latch to demultiplex the bus such that address and data are presented synchronously to the logic analyzer. When latching the bus, it is necessary to check setup and hold times of

*For more detailed discussion of these two architectures refer back to the main text, "Minicomputer Architectures."

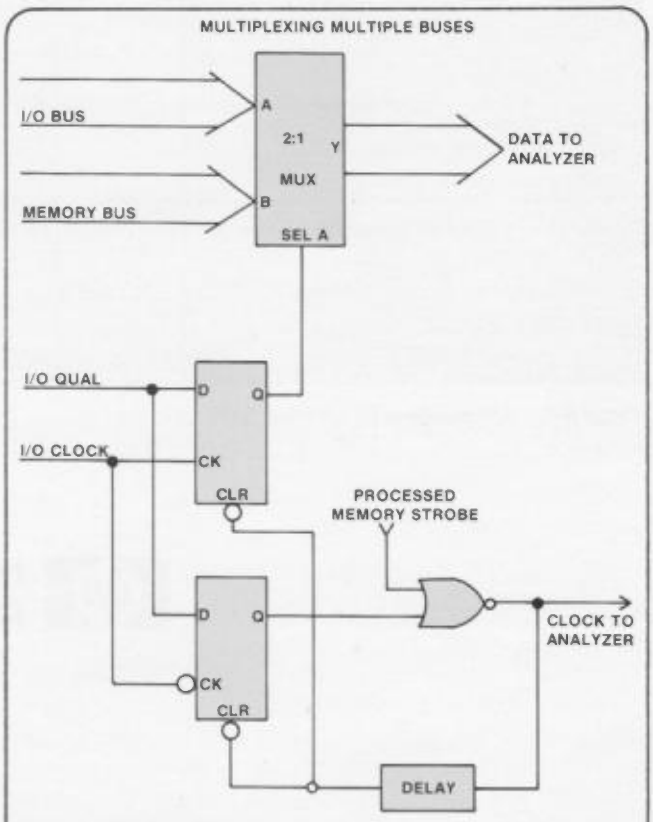


Figure A-4. Systems which operate synchronously often use two buses, a memory bus and a slower I/O bus.

the bus with respect to the address clock and the latch.

The minicomputer interface should ideally present a load to the bus that is less than, or equal to, the manufacturer's recommended specifications, both DC and AC. The DC input requirements of bus receivers are given in the IC manufacturer's data sheets. AC loading specs imply that the bus receivers should be located as close as possible to the edge connector to minimize stray capacitance.

There are several additional requirements of the interface for the logic analyzer. First, good probe grounding is necessary to maintain data integrity from the interface to the probe. This entails separate grounds for each probe brought back to the minicomputer bus for limiting ground loop problems. Analyzers also have specifications for minimum clock pulse width and minimum clock period. Also, data presented to the probes has a setup and hold time with respect to the clock. Table A-1 lists some of these specifications for HP's family of logic analyzers.

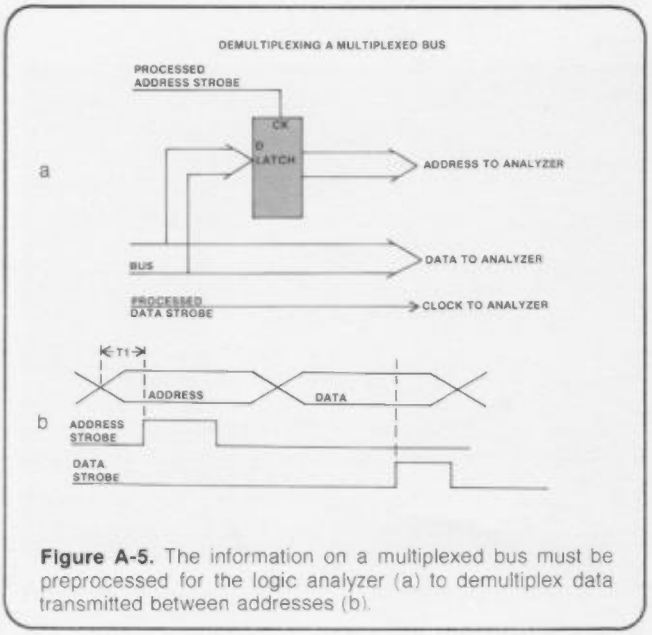


Figure A-5. The information on a multiplexed bus must be preprocessed for the logic analyzer (a) to demultiplex data transmitted between addresses (b).

	1600A	1600S	1602A	1607A	1611A Opt. 001	1615A	1610A	1610B
Data Inputs	16	32	16	16	36	24	32	32
Internal Store	16	16	64	16	64	256	64	64
Qualifiers	2	4	2	2	15	6	32	36
Sequential Triggers	—	1	—	—	1	1	7	7
Max Clock (MHz)	20	20	10	20	2.8	20	10	10
Compare	Stop ≠	Stop ≠	—	—	—	—	Stop $\overline{=}$ ≠	Stop $\overline{=}$ ≠
Number of Clocks	1	2	1	1	7	1	1	3
Data Setup time (ns)	20	20	35	20	80	20	20	20
Data Hold time (ns)	0	0	0	0	0	0	0	0
Clock Min Pulse Width (ns)	20	20	25	20	30	20	20	20

Table A-1. Specifications for HP Logic Analyzers.

W1QG

HEWLETT PACKARD

HP APPLICATION NOTES - DATA DOMAIN MEASUREMENTS

167-4 Engineering in the data domain calls for a new kind of digital instrument. (Reprinted from Electronics Magazine.)
167-5 Troubleshooting in the data domain is simplified by logic analyzers. (Reprinted from Electronics Magazine.)
167-6 Mapping, a dynamic display of digital system operation.
167-7 Supplementary data from map displays without changing probes.
167-9 Functional analysis of the Motorola M6800 microprocessor system.
167-11 Functional analysis of Intel 8008 microprocessor systems.
167-12 Functional analysis of Fairchild F8 microprocessor systems.
167-12A Functional analysis of MOSTEK F8 microprocessor systems.
167-13 The role of logic state analyzers in microprocessor based designs.
167-14 Functional analysis of 8080 microprocessor systems.
167-15 Functional analysis of Intel 4004 microprocessor systems.
167-16 Functional analysis of Intel 4040 microprocessor systems.
167-17 Functional analysis of National IMP microprocessor systems.
167-18 Functional analysis of National Semiconductor SC/MP microprocessor systems.
167-19 Systematic "turn-on" of microprocessor systems using logic state analyzers.
233-1 Functional analysis of Signetics 2650 microprocessor systems using the 1610A.
233-2 Functional analysis of TMS 9900 microprocessor systems using the 1610A.
233-3 Functional analysis of Z80 microprocessor systems using the 1610A.
233-4 Functional analysis of 8080 microprocessor systems using the 1610A.
233-5 Functional analysis of 6800 microprocessor systems using the 1610A.
260-1 Understanding Hewlett-Packard's Model 1615A Logic Analyzer.
275 Symptomatic troubleshooting of computer networks with HP 1640A.
275-1 Using the HP 1640A Serial Data Analyzer with the Epitape Recorder.
275-2 Using the HP 1640A Serial Data Analyzer with the Spectron Corp.T-511 Recorder.
280-1 Making Complex Measurements with the HP Model 1602A Logic State Analyzer.
280-2 Monitoring the IEEE-488 Bus with the 1602A Logic State Analyzer.
280-3 The 1602A Logic State Analyzer as an Automatic Test Instrument.
280-4 Using 1602A's for measurements on wide buses in manual and automatic modes.
292 Minicomputer analysis techniques using logic analyzers.
293 Functional analysis of microprocessor systems with the 1611A Opt 001 General Purpose Module.

For more information, call your local HP Sales Office or nearest Regional Office: Eastern (301) 258-2000; Midwestern (312) 255-9800; Southern (404) 955-1500; Western (213) 877-1282; Canadian (416) 678-9430. Ask the operator for Instrument Sales. Or, Write: Hewlett-Packard, 1501 Page Mill Road, Palo Alto, CA 94304. In Europe: Hewlett-Packard S.A., 7, rue du Bois-du-Lan, P.O. Box CH-1217 Meyrin 2, Geneva, Switzerland. In Japan: Yokogawa-Hewlett-Packard Ltd., 29-21, Takaido-Higashi 3-chome, Suginami-ku, Tokyo, 168.

5953-2704 AUG, 1979 **PRINTED IN U.S.A.**