Tektronix®

8500

MODULAR MDL SERIES

TRIGGER TRACE ANALYZER

USERS MANUAL

Tektronix®
COMMITTED TO EXCELLENCE

This manual supports the following
TEKTRONIX product:

8550F03

PLEASE CHECK FOR CHANGE INFORMATION AT THE REAR OF THIS MANUAL.

8500
MODULAR MDL SERIES
TRIGGER TRACE ANALYZER
USERS MANUAL

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

070-3760-01
Product Group 61

Serial Number _____

First Printing OCT 1981
Revised JAN 1984

LIMITED RIGHTS LEGEND

Software License No._____

Contractor: Tektronix, Inc.
Explanation of Limited Rights Data Identification Method
Used: Entire document subject to limited rights.

Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and /or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

PREFACE

ABOUT THIS MANUAL

The Trigger Trace Analyzer (TTA) is an option for a TEKTRONIX microcomputer development system. This manual provides information on the functions and usage of the TTA.

The Trigger Trace Analyzer Users Manual is divided into five sections, as follows:

Section 1	This Learning Guide contains general information about this instrument as well as examples of basic operating techniques.
Section 2	This Command Dictionary provides information about the commands that govern the operation of the TTA.
Section 3	This section (TTA Applications) contains examples that suggest how the TTA is used in typical design applications.
Section 4	This section (Technical Notes) provides information on TTA abilities, restrictions, and signal timing.
Section 5	This section (The Data Acquisition Interface) provides information on the function and use of the Data Acquisition Interface, an accessory to the TTA.

CHANGE INFORMATION

Change notices are issued by Tektronix, Inc., to document changes to the manual after it has been published. Change information is located at the back of this manual, following the yellow tab marked "CHANGE INFORMATION". When you receive this manual, you should enter any change information into the body of the manual, according to instructions on the change information page.

REVISION HISTORY

As this manual is revised and reprinted, revision history information is included on the text and diagram pages. Existing pages of manuls that have revised are indicated by REV and date (REV MAR 1983) at the bottom inside corner of the page. New pages added to an existing section, whether they contain old, new, or revised information, contain the word 'ADD' alongside the revision date (ADD MAR 1983).

DOCUMENTATION OVERVIEW

Support documentation for TEKTRONIX microcomputer development system consists of two groups of manuals: users manuals and service manuals.

Service manuals provide the information necessary to perform system testing, to isolate hardware problems, and to repair system components. Service manuals are identified by their blue covers and may be purchased from Tektronix as optional accessories.

The Trigger Trace Analyzer Installation Manual, although technically a service manual, is provided as a standard accessory to the TTA. This manual provides information on how to install the TTA.

CAUTION

The TTA may be factory-installed, if ordered with your microcomputer development system, or obtained as a field option. If your TTA has not been factory-installed, do not try to install it yourself. Since Tektronix, Inc., is not obligated to repair damage resulting from attempts by unauthorized personnel to install this product, you should have a Tektronix Field Service Specialist install your TTA. Please call the nearest Tektronix Field Service Office for installation.

User manuals describe how to operate a TEKTRONIX microcomputer development system and its associated options. User manuals are identified by their grey covers and are provided as a standard accessory to an instrument. The following manuals provide information on the use of Tektronix development systems:

- 8550 Microcomputer Development Lab System Users Manual
- 8540 Integration Unit System Users Manual

The Trigger Trace Analyzer Users manual is provided as a standard accessory to the Trigger Trace Analyzer.

To derive the greatest benefit from this manual, you should be familiar with your TEKTRONIX development system, as described in its system users manual.

CONTENTS

CONTENTS (CONT)

ILLUSTRATIONS

TABLES

OPERATORS SAFETY SUMMARY

The general safety information in this part of the summary is for both operating and servicing personnel. Specific warnings and cautions will be found throughout the manual where they apply, but may not appear in this summary.

TERMS

In This Manual

CAUTION statements identify conditions or practices that could result in damage to the equipment or other property.

WARNING statements identify conditions or practices that could result in personal injury or loss of life.

As Marked on Equipment

CAUTION indicates a personal injury hazard not immediately accessible as one reads the marking, or a hazard to property including the equipment itself.

DANGER indicates a personal injury hazard immediately accessible as one reads the marking.

SYMBOLS

As Marked on Equipment


DANGER high voltage.


Protective ground (earth) terminal.


ATTENTION - Refer to manual.

SAFETY PRECAUTIONS

Grounding the Product

This product is grounded through grounding conductors in the interconnecting cables. To avoid electrical shock, plug the supporting system's power cord into a properly wired receptacle. A protective ground connection by way of the grounding conductor in the power cord is essential for safe operation.

Use the Proper Power Cord

Use only the power cord and connector specified for your product. Use only a power cord that is in good condition. Refer cord and connector changes to qualified service personnel.

Use the Proper Fuse

To avoid fire hazard, use only the fuse specified in the parts list for your product. Be sure the fuse is identical in type, voltage rating, and current rating.

Refer fuse replacement to qualified service personnel.

Do Not Operate in Explosive Atmospheres

To avoid explosion, do not operate this product in an atmosphere of explosive gases unless it has been specifically certified for such operation.

Do Not Remove Covers or Panels

To avoid personal injury, do not remove the product covers or panels. Do not operate the product without the covers and panels properly installed.

Section 1

LEARNING GUIDE

INTRODUCTION

This Learning Guide provides an overview of the features and functions of the Trigger Trace Analyzer (TTA). It also contains a Demonstration Run that you can use to obtain hands-on experience. The Learning Guide is divided into the following parts:

- Overview of TTA Operation
- Physical Parts of the TTA
- Demonstration Run

SPECIFICATIONS, INSTALLATION, CONFIGURATION, AND VERIFICATION

For information on how to install your TTA hardware, and for product specifications, refer to your Trigger Trace Analyzer Installation Service Manual.

OVERVIEW OF TTA OPERATION

The Trigger Trace Analyzer (TTA) is a real-time debugging tool. When used in conjunction with a TEKTRONIX 8500-series development system, the TTA will monitor each bus transaction of a program as that program is executed. Before you execute a program, you can program the TTA to perform several kinds of operations on the bus transactions that it monitors. These operations include the TTA's ability to:

- recognize one bus transaction as an event.
- recognize consecutive bus transactions as an event.
- count the iterations of an event.
- measure the interval (in real time or clock cycles) between events.

- halt program execution when an event occurs.
- display information about a bus transaction that the TTA recognizes as an event, without halting program execution.
- capture and store information about up to 255 bus transactions that occur before, during, or following an event.

TRIGGER CHANNELS

The TTA has four trigger channels. Each channel consists of an event comparator and a programmable counter. Within the trigger channel, an event signal (from an event comparator) and a counter signal (from a programmable counter) are ANDed together to create a trigger signal.

Event Comparators

Essentially, an event comparator functions as a word recognizer. An event is some unique set of signal states that occurs during a single bus transaction. Each of the TTA's four event comparators may be programmed to detect a specific event.

The following signals are input to each event comparator and may be used to define an event:

- 24 address bus signals
- 16 data bus signals
- 8 probe signals
- 4 counter output signals
- 1 event qualifier signal
- 11 emulator-dependent bus control signals

You can define any combination of these signals as an event. That is, an event is actually the "sum" of six possible event definitions. For example, if an address and a byte of data were both defined as an event for the same event comparator, the TTA would only recognize a bus transaction that contained both that address and that byte of data as an event.

Programmable Counters

Each trigger channel has a programmable counter. The outputs from these counters act to qualify the event signal within a trigger channel, or work independently of the event comparator to create a trigger signal for some counter operation.

Trigger Signals

Trigger signals 1—3 (SN B030000 and up) or trigger signals 1—4 (SN B029999 and below) are available via a BNC connector on the Data Acquisition Interface. These external signals may be used to trigger some device external to the development system, such as a logic analyzer. In addition, the trigger signal is used internally to support the following functions:

- triggering a breakpoint
- clocking a programmable counter
- gating a programmable counter

Beginning with SN B030000 and up, an Acquisition Clock Output is available via a BNC connector, marked ACQ CLK OUT on the Data Acquisition Interface. This clock signal occurs when the TTA has acquired a sample of address, data, or other information. The signal is intended for operation together with a TEKTRONIX DAS 9100-Series System.

ACQUISITION MEMORY

Acquisition Memory captures and stores information about the bus transactions of a program. This information may be recorded for selected bus transactions, or (at default) for the last 255 bus transactions that occurred. For each bus transaction, the recorded information consists of:

- an address
- data
- an opcode mnemonic
- the states of the eight Data Acquisition Probe signals
- symbols representing the type of I/O operation that occurred

Refer to the acq and disp commands, within the Command Dictionary, for information on accessing the TTA's Acquisition Memory.

TTA STATUS DISPLAY

The TTA status display provides information about the current programming of the TTA. This display is called up with the ts command and will indicate:

- current event definitions for each trigger channel
- currently selected cons (consecutive) command parameters
- current counter programming for each trigger channel
- currently selected bre (breakpoint) command parameters
- currently selected acq (acquire) command parameters

When learning to use the TTA, you should use the ts command frequently, before and after entering a command, to observe the effect of that command upon the TTA status display and current TTA programming.

PHYSICAL PARTS OF THE TTA

The TTA option consists of the following subassemblies:

- two TTA circuit boards
- two TTA interconnect cables
- the data acquisition interface unit
- the data acquisition probe

The two TTA circuit boards plug into the Main Interconnect Board of a TEKTRONIX 8500-series development system. These circuit boards contain all of the circuitry associated with the functions and features of the TTA commands.

The TTA Interconnect Cables fit across the two top edge connectors on TTA Circuit Board #1 and TTA Circuit Board #2. The TTA Interconnect Cables simply make an electrical connection between these two circuit boards.

The Data Acquisition Interface is installed within the rear panel of your development system. This subassembly provides an output for trigger signals

and allows you to interface the Data Acquisition Probe and external instruments to the rest of TTA circuitry. Refer to Section 5 for a description of the Data Acquisition Interface and its operation.

The Data Acquisition Probe allows you to monitor an external clock and eight external data channels. The Data Acquisition Probe attaches to the Data Acquisition Interface via a connector at the end of a 25-line cable. This connector fits into a socket on the control panel of the Data Acquisition Interface.

DEMONSTRATION RUN

This demonstration provides basic information you will need to begin using your TTA. Since the TTA may be used with more than one TEKTRONIX 8500-series development system, initial start-up procedures will vary. Thus, this demonstration run will begin at a point that is common to all compatible systems. However, it is assumed that your development system, the TTA, and your system terminal have been unpacked, installed, and tested by qualified personnel.

This Demonstration Run can be completed in approximately twenty minutes.

GETTING STARTED

Since the TTA is emulator-dependent, a sample program for the 8085A emulator is used here as an example. If you are using an emulator other than the 8085A, refer to the Emulator Specifics section of your System Users Manual for a sample program that is parallel to this one.

To complete this demonstration run, you will need a development system with the following options installed:

- Trigger Trace Analyzer
- 8085A Emulator Processor
- 8085A emulator software

THE SAMPLE PROGRAM

The 8085A sample program begins at location 0100 in program memory. This program sums five numbers (1, 2, 3, 4, and 5) and places the result in the 8085A's accumulator. The last instruction of this program initiates an SVC and returns control of the system to the user. Display 1-1 illustrates a program listing for the 8085A sample program.

```
ADDRESS  DATA      MNEMONIC      Explanation

 0040    00
 0041    42                      Set up service call-
 0042    1A                      for exit.

 0100    210005    LXI H,500     Set table pointer.
 0103    0605      MVI B,5       Set pass counter.
 0105    AF        XRA A         Clear accumulator.
 0106    86        ADD M         Add byte from table.
 0107    23        INX H         Point to next byte.
 0108    05        DCR B         Decrement pass counter.
 0109    C20601    JNZ 0106      Loop if not fifth pass.
 010C    D3F7      OUT F7        If fifth pass then exit.

 0500    01                      First location in data table.
 0501    02                      Second location in data table.
 0502    03                      Third location in data table.
 0503    04                      Fourth location in data table.
 0504    05                      Fifth location in data table.
```

Display 1-1
Sample Program Listing

ENTERING THE SAMPLE PROGRAM

The following steps are used to enter the 8085A sample program into your system's program memory.

NOTE

> Before you enter this sample program, you must first start up your development system and system terminal, as explained in your System Users Manual.

After the appropriate "boot-up" message for your system has appeared, and the system has displayed a prompt sign (>), select the 8085 emulator, with the following command:

> sel 8085 <CR>

Fill the program memory that you will be using with zeros, with the following command:

> f 0 600 00 <CR>

Patch the 8085A sample program's SVC set-up information into program memory, with the following command:

> p 040 00 42 1A <CR>

Dump the program memory that you just patched, with the following command:

> d 040 <CR>

Compare the display on your terminal with the one shown here, and verify that program memory was patched correctly:

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040  00 42 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 .B..............
```

Patch the main body of the 8085A sample program into program memory, with the following command:

> p 100 21 00 05 06 05 AF 86 23 05 C2 06 01 D3 F7 <CR>

Dump the program memory that you just patched, with the following command:

> d 100 <CR>

Compare the display on your terminal with the one shown here and verify that program memory was patched correctly.

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000100  21 00 05 06 05 AF 86 23 05 C2 06 01 D3 F7 00 00 !..............
```

Patch the five bytes that the 8085A sample program will sum into program memory, with the following command:

> p 500 01 02 03 04 05 <CR>

Dump the program memory that you just patched, with the following command:

> d 500 <CR>

Compare the display on your terminal with the one shown here, and verify that program memory was patched correctly:

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000500 01 02 03 04 05 00 00 00 00 00 00 00 00 00 00 00 ................
```

At this point, you have entered the 8085A sample program into your system's program memory. To verify that this program was entered correctly, execute the sample program, with the following command:

> g 100 <CR>

The following information is displayed on your system terminal:

```
LOC  INST   MNEM  OPER     SP     F   A   B  C  D  E  H  L  IM SOD
010F 00     NOP            0000   54  0F  00 00 00 00 05 05  00  0
010F   <BREAK   >
```

The "A" in the top row of this display represents the accumulator for the 8085A Emulator. The "0F" below the "A" indicates that the content of the accumulator is 0F hexadecimal (15 decimal), the sum of the five numbers (1+2+3+4+5). Essentially, this display indicates that the sample program operated as intended.

USING THE TTA

The following examples illustrate how to use the TTA. These examples are organized in a sequence that gradually introduces you to TTA operations. Each example describes a TTA operation and tells you how to perform that operation on the 8085A sample program. For information about any command used within the examples, refer to the Command Dictionary, in Section 2 of this manual. For examples of more complex TTA operations, or examples of typical design applications for the TTA, refer to Section 3 of this manual.

The Display Command

The disp (display) command is used to call up the window of data that is stored in the TTA's Acquisition Memory. Each time a program is executed, some information is stored in Acquisition Memory. To demonstrate this, enter:

```
> g 100 <CR>
```

The following information is displayed on your system terminal:

```
LOC  INST   MNEM  OPER     SP     F   A   B  C  D  E  H  L  IM SOD
010F 00     NOP            0000   54  0F  00 00 00 00 05 05  00  0
010F   <BREAK   >
```

This display indicates that the program was executed. Now, display the information that was stored during the execution of this program, by entering:

```
> disp <CR>
```

Display 1-2 shows how information is displayed on your terminal:

```
 ADDR  DATA  MNEMONIC        7-PROBE-0    BUS
000100  21   LXI    H       0000 0000   M RD F
000101  00                  0000 0000   M RD
000102  05                  0000 0000   M RD
000103  06   MVI    B       0000 0000   M RD F
000104  05                  0000 0000   M RD
000105  AF   XRA    A       0000 0000   M RD F
000106  86   ADD    M       0000 0000   M RD F
000500  01                  0000 0000   M RD
000107  23   INX    H       0000 0000   M RD F
000108  05   DCR    B       0000 0000   M RD F
000109  C2   JNZ             0000 0000   M RD F
00010A  06                  0000 0000   M RD
00010B  01                  0000 0000   M RD
000106  86   ADD    M       0000 0000   M RD F
000501  02                  0000 0000   M RD
000107  23   INX    H       0000 0000   M RD F
000108  05   DCR    B       0000 0000   M RD F
000109  C2   JNZ             0000 0000   M RD F
00010A  06                  0000 0000   M RD
00010B  01                  0000 0000   M RD
000106  86   ADD    M       0000 0000   M RD F
000502  03                  0000 0000   M RD
000107  23   INX    H       0000 0000   M RD F
000108  05   DCR    B       0000 0000   M RD F
000109  C2   JNZ             0000 0000   M RD F
00010A  06                  0000 0000   M RD
00010B  01                  0000 0000   M RD
000106  86   ADD    M       0000 0000   M RD F
000503  04                  0000 0000   M RD
000107  23   INX    H       0000 0000   M RD F

 ADDR  DATA  MNEMONIC        7-PROBE-0    BUS
000108  05   DCR    B       0000 0000   M RD F
000109  C2   JNZ             0000 0000   M RD F
00010A  06                  0000 0000   M RD
00010B  01                  0000 0000   M RD
000106  86   ADD    M       0000 0000   M RD F
000504  05                  0000 0000   M RD
000107  23   INX    H       0000 0000   M RD F
000108  05   DCR    B       0000 0000   M RD F
000109  C2   JNZ             0000 0000   M RD F
00010A  06                  0000 0000   M RD
00010C  D3   OUT             0000 0000   M RD F
00010D  F7                  0000 0000   M RD
00F7F7  0F                  0000 0000   I WT
00010F  00   NOP             0000 0000   M RD F
```

Display 1-2

Display 1-2 illustrates the information that is stored within the TTA's Acquisition Memory. This information pertains to each recorded bus transaction and includes:

- an address
- data
- an opcode mnemonic
- the states of the eight Data Acquisition Probe signals
- symbols representing the type of I/O operation that occurred

Defining a Simple Event

The ad command is used to define an address or a range of addresses as an event. For our first example, let's define any access of a single address as an event. To do this, we select address 0106 and trigger channel 1. Enter the following command line to define any access of location 0106 as event 1:

> ad 1 106 <CR>

Setting a Breakpoint

The bre command is used to set a breakpoint for a trigger channel. Now that you have defined event 1, you can tell the TTA to halt program execution when event 1 occurs, and return control to the system terminal. Enter:

> bre 1 <CR>

> g 100 <CR>

The following display appears on your terminal:

```
LOC  INST   MNEM  OPER      SP     F   A  B  C  D  E  H  L  IM  SOD
0106 86     ADD   M         0000   00  01 05 00 00 00 05 00  07   0
0106    <BREAK    TRIG1>
```

This display indicates that a breakpoint occurred when location 0106 was accessed and that this breakpoint was associated with trigger channel 1. Since the trigger signal for trigger channel 1 is enabled by event 1, we know that event 1 occurred.

Selecting the Breakpoint Option

The cont parameter, entered within a bre command line, selects the breakpoint option. This option allows you to monitor a program without interrupting the execution of that program. In the preceding example, you set a breakpoint for event 1. When this event occurred, the sample program was interrupted. Now, use the breakpoint option to display every occurrence of event 1, without interrupting the program. Enter:

> bre 1 cont <CR>

> g 100 <CR>

The following display appears on your terminal:

```
LOC  INST   MNEM  OPER      SP     F   A  B  C  D  E  H  L  IM  SOD
0106 86     ADD   M         0000   00  01 05 00 00 00 05 00 07   0
0106 86     ADD   M         0000   04  03 04 00 00 00 05 01 07   0
0106 86     ADD   M         0000   04  06 03 00 00 00 05 02 07   0
0106 86     ADD   M         0000   04  06 02 00 00 00 05 03 07   0
0106 86     ADD   M         0000   04  0B 01 00 00 00 05 04 07   0
010F 00     NOP             0000   54  0B 00 00 00 00 05 05 07   0
010F   <BREAK   >
```

This display indicates that address 0106 was accessed five times during the execution of the sample program. The last line of this display shows that a system breakpoint occurred at location 010F (the SVC routine) and that the sample program ran through to its conclusion.

Eliminating Unwanted Parameters

At this point, you've defined an event, set a breakpoint, and used the breakpoint option. In the following examples, you will expand upon these functions. First, however, you must eliminate the parameters that you have already selected.

To eliminate the event you defined as address 0106, enter:

> ad 1 clr <CR>

To eliminate the breakpoint (or the breakpoint option in this case), enter:

> bre 1 clr <CR>

DEBUGGING A PROGRAM

Thus far, we've looked at some independent TTA operations. Now, we'll use these same operations and others while debugging a program. However, since the program now executes correctly, we must first introduce an error into the sample program.

Introducing an Error

The sample program adds five numbers and stores the result in the emulator's accumulator. To introduce a deliberate error into this program, let's change one of the five numbers (04) to another number (00). Enter:

> ex 503 <CR>

When 00000503=04 is displayed, enter 00

When 00000504=05 is displayed, enter a carriage return.

The sample program now contains an error. You changed the value that was stored at location 503 from 04 to 00. To demonstrate this, enter:

> g 100 <CR>

The following display appears on your terminal:

```
LOC  INST   MNEM  OPER      SP     F   A  B  C  D  E  H  L  IM  SOD
010F 00     NOP             0000  54  0B 00 00 00 00 05 05  07   0
010F   <BREAK   >
```

Note that the accumulator now contains the hexadecimal value 0B (11 decimal) rather than 0F (15 decimal).

Finding the Error

Let's make some assumptions: The sample program is a subroutine within a very large program. You executed the program and detected an error. When you reviewed the program listing, the cause of the error was not apparent. You could use your system's tra (trace) command and step through the entire program, but this would be very time-consuming.

Recall that the program adds five numbers, and that these five numbers are contained in separate memory locations. This means that the program must execute the 8085A ADD M instruction five times. Therefore, the first step in debugging this program might be to verify that each of these five ADD M instructions were executed.

First, define the ADD M instruction as an event. (The 8085A's ADD M instruction is opcode 86.) Enter:

> data 1 86 <CR>

At this point, we could set a breakpoint for trigger channel 1. However, the breakpoint would halt the program on the first occurrence of this event. Since you want to monitor all occurrences of event 1 (the ADD M instruction), use the breakpoint option. Enter:

> bre 1 cont <CR>

Now, execute the program. Enter:

> g 100 <CR>

The following display appears on your terminal:

```
LOC  INST   MNEM  OPER      SP     F   A  B  C  D  E  H  L  IM  SOD
0106 86     ADD   M         0000   00  01 05 00 00 00 05 00  07   0
0106 86     ADD   M         0000   04  03 04 00 00 00 05 01  07   0
0106 86     ADD   M         0000   04  06 03 00 00 00 05 02  07   0
0106 86     ADD   M         0000   04  06 02 00 00 00 05 03  07   0
0106 86     ADD   M         0000   04  0B 01 00 00 00 05 04  07   0
010F 00     NOP             0000   54  0B 00 00 00 00 05 05  07   0
010F    <BREAK  >
```

This display indicates that the program did execute the ADD M instruction five times. However, note that the contents of the accumulator (register A) did not change following the fourth ADD operation. This suggests that the number stored at the fourth memory location, and accessed by the fourth ADD M instruction, is not correct.

Now, let's assume that the error was not entered deliberately. You suspect that one of the data locations contains an error, but how do you determine which one? One method available with the TTA is to selectively acquire the data accessed from each of the five memory locations. First, eliminate the parameters that were previously set:

> data 1 clr <CR>

> bre 1 clr <CR>

These command lines clear the event and breakpoint option that you used to monitor the five ADD M instructions.

Now, we'll select events and cause a window of data associated with these events to be stored in the TTA's Acquisition Memory.

First, define event 3 as the ADD M instruction. Enter:

> data 3 86 <CR>

Next, define event 4 as any memory-read operation:

> bus 4 m rd <CR>

Now, qualify event 4 by making it dependent upon the occurrence of event 3 on a preceding cycle.

> cons cyc 34 <CR>

Finally, enter the acq ev4 command. With this command, only bus transactions recognized as event 4, as you have just defined it, will be recorded in Acquisition Memory. Enter:

> acq ev4 <CR>

At this point, you have programmed the TTA to store information about a bus transaction (defined as event 4) only if it follows the occurrence of event 3 (an ADD M instruction).

To obtain the result of the operation, enter:

> g 100 <CR>

The following display appears on your terminal:

```
LOC  INST   MNEM  OPER      SP     F   A  B  C  D  E  H  L  IM  SOD
010F 00     NOP             0000   54  0B 00 00 00 00 05 05  07   0
010F    <BREAK   >
```

The sample program has been executed and the TTA has stored information within Acquisition Memory. To view this window of data, enter:

> disp <CR>

The following display appears on your terminal:

```
 ADDR  DATA  MNEMONIC        7—PROBE—0     BUS
000106  86   ADD   M         0000 0000   M RD F
000500  01                   0000 0000   M RD
000106  86   ADD   M         0000 0000   M RD F
000501  02                   0000 0000   M RD
000106  86   ADD   M         0000 0000   M RD F
000502  03                   0000 0000   M RD
000106  86   ADD   M         0000 0000   M RD F
000503  00                   0000 0000   M RD
000106  86   ADD   M         0000 0000   M RDF
000504  05                   0000 0000   M RD
```

This display shows the contents of each of the five memory locations that the 8085A ADD instruction operated upon. Note the error at location 0503. If you were debugging this program, you could now use your system's e (exam) or p (patch) command to correct the error.

SUMMARY OF THE TTA DEMONSTRATION RUN

In this demonstration, you loaded the sample program, monitored it with the TTA, and used the TTA to detect an error. Now, review the TTA commands that you used to perform these operations:

- disp -- displays the contents of Acquisition Memory
- ad -- defines an address as an event
- bre -- sets a breakpoint
- data -- defines a byte of data as an event
- bus -- defines the states of bus control signals as an event
- cons -- links the occurrence of events
- acq -- selects the information to be placed in Acquisition Memory

Section 2

COMMAND DICTIONARY

INTRODUCTION

This Command Dictionary describes each of the TTA commands. The section is divided into the following parts:

- Command Syntax. Describes the notation used within the syntax blocks of this dictionary.
- Command Parameters. Describes the notation used within the explanation of a command's parameters.
- The TTA Commands. Describes each TTA command individually. The commands are described in alphabetical order.

COMMAND SYNTAX

The description of each command includes a syntax block. The syntax block illustrates one or more command line formats. Each command line format contains the command name and indicates those parameters that can or must be included in the command line. Figure 2-1 is an example of a syntax block.

SYNTAX

sample

sample all clr

sample [**-n**] [**-s** / **-c**] (**1** / **2** / **3** / **4**) {**value**}

Fig. 2-1 Sample Syntax Block

In Fig. 2-1, three command line formats are shown. Any of the three formats may be used with this command.

In the first command line format, the command name (sample) is used without parameters.

In the second command line format, the "all clr" parameter must be included. In the third command line format, the command name and a parameter from within each of two braces (a number and a value) must be included to call up some function. In addition, the optional parameters "-s" or "-c" may be used, with or without the optional parameter "-n".

COMMAND PARAMETERS

Parameters specify or modify the way in which a command is used. The function of each parameter is described for every command within this command dictionary. The following rules apply to parameters within the Syntax blocks.

- Parameters and special characters that appear in boldface must be entered exactly as they appear within a syntax block.
- Parameters not in boldface indicate a variable parameter. The replacements for these variables are listed in the parameter description that follows each command explanation.
- A single parameter that is required appears in the syntax block without braces or brackets.
- If there is a choice of required parameters, each parameter appears in the syntax block stacked within the braces {}.
- Optional parameters appear in the syntax block within brackets []. If there is a choice of optional parameters, they will be stacked within the brackets.
- If more than one of the parameters stacked within braces or brackets can be used within a command line, these braces or brackets will be followed by an ellipsis [...].
- Command modifiers are indicated by a single letter preceded by a hyphen (-). These parameters are optional and will appear in brackets.
- Where permitted, the short form of a command or parameter is underlined.

THE TTA COMMANDS

There are 14 TTA commands. In this section, the commands are listed in alphabetical order. Each command listing contains a syntax block, an explanation of the command, an explanation of the parameters that can be used with the command, and examples of command usage. The TTA commands are:

- acq - determines what is stored in Acquisition Memory
- ad - defines address parameters for events
- bre - controls the breakpoint for each trigger
- bus - defines bus signal parameters for events
- cons - defines consecutive events
- cou - programs the TTA's four counters
- ctr - defines the counter outputs for events
- data - defines data parameters for events
- disp - displays acquisition memory
- eve - defines all parameters for an event
- pro - defines the probe parameters for events
- qua - defines external event qualifiers for events
- tclr - returns the TTA parameters to default status
- ts - displays current TTA programming status

SYNTAX

acq

```
     {ev4}  [for expression source                    ]
acq  {all}  [for expression source aftertrig4]
```

EXPLANATION

The acq (acquire) command determines which bus transactions are to be stored in the TTA's Acquisition Memory. This command selects some number of transactions to be stored or the kind of transaction to be stored. If no parameters are selected, the most recent 255 bus transactions of a program will be stored in Acquisition Memory.

PARAMETERS

acq — When you enter the acq command without parameters, the currently selected acq parameters are displayed on the system terminal.

all — Specifies that all bus transactions will be stored in the TTA's Acquisition Memory as they occur.

ev4 — Specifies that only those bus transactions defined as event 4 will be stored in Acquisition Memory.

for expression source — Causes Acquisition Memory to stop storing bus transactions at some point other than the end of a program.

The expression must evaluate to some number from 1 to 65535 (decimal). The numeric value for this expression is assumed to be decimal, unless another format is specified. Refer to your system's user manual for information on other formats for representing numeric values.

The source portion of this parameter identifies a specific kind of bus transaction. Table 2-1 lists valid options for the acq command's source parameters. Note that these source parameters are also used with the cou command.

aftertrig4 — Disables the counting of the source until trigger 4 occurs. If this option is specified, the expression is limited to some number from 2 to 65535 (decimal).

Table 2-1
Acquisition Source Parameters

Parameters	Signal Source for Acquisition Counter
s=200nsec	200 nsec clock
s=2usec	2 usec clock
s=20usec	20 usec clock
s=200usec	200 usec clock
s=2msec	2 msec clock
s=ev1	The event signal for channel 1 ANDed with cyc
s=ev2	The event signal for channel 2 ANDed with cyc
s=ev3	The event signal for channel 3 ANDed with cyc
s=ev4	The event signal for channel 2 ANDed with cyc
s=trig1	The trigger signal for channel 1
s=trig2	The trigger signal for channel 2
s=trig3	The trigger signal for channel 3
s=trig4	The trigger signal for channel 4
s=acq	The INCR PTR signal (*a)
s=cyc	The SLV OPREQ signal (*b)
s=emuclk	The emulator's clock signal
s=qua	The event qualifier signal (*c)

(*a) The INCR PTR signal occurs each time the TTA stores a bus transaction.

(*b) The SLV OPREQ signal occurs on each emulator cycle.

(*c) The event qualifier signal comes from the Data Acquisition Probe.

EXAMPLES

Store up to 255 bus transactions that precede the end of the program or a breakpoint, with the following command line:

> acq all <CR>

Store only those bus transactions that satisfy the definition of event 4, with the following command line:

> acq ev4 <CR>

Store only the first 10 bus transactions, with the following command line:

> acq all for 10 cyc <CR>

Store only the first 10 bus transactions that satisfy the definition of event 4, with the following command line:

> acq ev4 for 10 acq <CR>

Store all bus transactions until the occurrence of the 10th cycle after the occurrence of event 4, with the following command line:

> acq all for 10 cyc aftertrig4 <CR>

SYNTAX

ad all clr

ad [**-c** | **-s**] (**4** | **3** | **2** | **1**) {**clr**}

ad [**-n**] [**-c** | **-s**] (**4** | **3** | **2** | **1**) {address address | address}

EXPLANATION

The ad (address) command defines an address or a range of addresses as an event or part of an event.

PARAMETERS

1 Specifies that the command line applies only to event 1.

2 Specifies that the command line applies only to event 2.

3 Specifies that the command line applies only to event 3.

4 Specifies that the command line applies only to event 4.

all Specifies that the command line applies equally to all four events.

clr Clears the current address definition for the indicated event(s).

-s Sets a breakpoint for the trigger channel associated with the indicated event. This breakpoint interrupts the program when the trigger channel's output signal is enabled, prints a trace line, and returns control to the operating system. The -s parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-c Selects the breakpoint "continue" option. This option sets a breakpoint for the trigger channel associated with the indicated event, interrupts the program when that trigger channel's output signals is enabled, and prints a trace line. However, unlike the "stop" option, the "continue" option returns control to the program rather than to the operating system. The -c parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-n Defines the event as any address other than the address specified by the address parameter(s).

address Specifies the address value (or range of values) to be detected. An address may be defined as a symbolic expression or an absolute address. An absolute address is assumed to be hexadecimal; however, any numeric format may be used if the rules appropriate for your system are observed. An "x" may be substituted for any digit within a legal numeric expression for a "don't care" situation. Note that the number of address lines, and legal addresses, are emulator-dependent.

EXAMPLES

Define event 1 as the absolute address 0500H, with the following command line:

> ad 1 500 <CR>

Define event 3 as a binary absolute address, with the following command line:

> ad 3 1010101010101111Y <CR>

Define event 2 as a range of addresses between 0500H and 0505H, with the following command line:

> ad 2 500 505 <CR>

Define event 2 as any address not within a range of addresses between 0500H and 0505H, with the following command line:

> ad -n 2 500 505 <CR>

Define event 3 as a hexadecimal address with "don't care" digits, with the following command line:

> ad 3 0X5XX <CR>

Define event 3 as a binary address with "don't care" digits, with the following command line:

> ad 3 0XXX1XX1X1XXXX1XXY <CR>

Clear the address parameters previously defined as event 4, with the following command line:

> ad 4 clr <CR>

Clear the address parameters previously defined for all four events, with the following command line:

> ad all clr <CR>

NOTE

Many processors have two-digit I/O ports that are seen as addresses by the emulator. The upper eight address lines `float' and may not always be 00. It is recommended that when using I/O ports, the unused address bits be padded with don't-cares. For example, I/O port 73 would be defined as an address event by entering 0XX73 as the address parameter. If you omit the don't-cares with I/O ports, the TTA software will assume leading 0's.

SYNTAX

bre

bre { **4** | **3** | **2** | **1** | **all** }

bre { **4** | **3** | **2** | **1** | **all** } { **-c** | **-s** | **cont** | **stop** | **clr** } **[...]**

EXPLANATION

The bre (breakpoint) command controls the effects of an event's trigger signal. For each trigger, this command can set a breakpoint, clear a breakpoint, enable the continue function, or disable the continue function. The breakpoint, if enabled, causes a program to halt execution when an event and its associated trigger signal occurs. The continue function, if enabled, causes a trace line to be displayed on the system terminal when an event and trigger signal occurs, and allows program execution to continue.

PARAMETERS

bre — When you enter the bre command without parameters, the currently selected bre parameters are displayed on the system terminal.

1 — Specifies that the command line applies only to trigger channel 1.

2 — Specifies that the command line applies only to trigger channel 2.

3 — Specifies that the command line applies only to trigger channel 3.

4 — Specifies that the command line applies only to trigger channel 4.

all — Specifies that the command line applies equally to all four trigger channels.

clr — Clears the current breakpoint or continue option previously selected for the indicated trigger channel(s).

stop (or -s) — Sets a breakpoint for the indicated trigger channel. This breakpoint interrupts the program when the trigger channel's output signal is enabled, prints a trace line, and returns control to the operating system. The stop parameter takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

cont (or -c) — Selects the breakpoint "continue" option. This option sets a breakpoint for the indicated trigger channel, interrupts the program when that trigger channel's output signals is enabled, and prints a trace line. However, unlike the "stop" option, the "continue" option returns control to the program rather than to the operating system. The -c parameter takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

... — Indicates that parameters in the preceding set of braces or brackets may be repeated in the command line.

EXAMPLES

Set a breakpoint for trigger channel 1, with the following command line:

> bre 1 stop <CR>

(or)

> bre 1 <CR>

Clear the breakpoint for trigger channel 1, with the following command line:

> bre 1 clr <CR>

Enable the continue function for trigger channel 2, with the following command line:

> bre 2 cont <CR>

Clear the breakpoints for all four trigger channels, with the following command line:

> bre all clr <CR>

Set breakpoints for trigger channels 1, 2, and 3, with the following command line:

> bre 1 stop 2 stop 3 stop <CR>

Set a breakpoint for trigger channel 2 and clear the breakpoints for trigger channels 1 and 3, with the following command line:

> bre 1 clr 2 stop 3 clr <CR>

SYNTAX

bus all clr

bus [**-s** / **-c**] (**4** / **3** / **2** / **1**) (**clr** / symbol) **[...]**

EXPLANATION

The bus command is used to define an event or part of an event as the assertion of one or more bus/control signals. Within the bus command line, each bus/control signal is represented by an emulator-dependent symbol. Since these symbol parameters are ANDed, an event occurs only when all of the selected signals are asserted. Refer to the Emulator Specifics section of your system users manual for a list of the symbols appropriate for your emulator.

PARAMETERS

1 Specifies that the command line applies only to event 1.

2 Specifies that the command line applies only to event 2.

3 Specifies that the command line applies only to event 3.

4 Specifies that the command line applies only to event 4.

all Specifies that the command line applies equally to all four events.

clr Clears the current bus definition for the indicated event(s).

-s Sets a breakpoint for the trigger channel associated with the indicated event. This breakpoint interrupts the program when the trigger channel's output signal is enabled, prints a trace line, and returns control to the operating system. The -s parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-c Selects the breakpoint "continue" option. This option sets a breakpoint for the trigger channel associated with the indicated event, interrupts the program when that trigger channel's output signals is enabled, and prints a trace line. However, unlike the "stop" option, the "continue" option returns control to the program rather than to the operating system. The -c parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

symbol A symbol that represents one of the emulator-dependent bus/control signals. Refer to the Emulator Specifics section of your system users manual for a list of those symbols that apply to your emulator.

... This parameter indicates that more than one symbol may be entered within one command line; selecting more than one bus/control signal as an event.

NOTE

The symbols used in the following examples represent the bus/control signals of the 8085A emulator. These symbols do not necessarily represent the bus/control signals for other emulators.

EXAMPLES

Define event 1 as a fetch cycle, with the following command line:

> bus 1 f <CR>

Define event 2 as a non-fetch cycle, with the following command line:

> bus 2 nf <CR>

Define event 3 as a memory write cycle, with the following command line:

> bus 3 m wt <CR>

Clear the bus command parameters for event 4, with the following command line:

> bus 4 clr <CR>

Clear the bus command parameters for all events, with the following command lines:

> bus all clr <CR>

SYNTAX

cons

cons clr

cons { **emu** / **fet** / **cyc** } {sequence} **[...]**

EXPLANATION

The cons (consecutive) command defines a sequence of consecutive events. Once a sequence is defined, only the last event in that sequence can enable its respective trigger signal. The events that precede the last event in a sequence act to qualify that last event. Note that all of the events within a sequence must occur on consecutive cycles of the specified type.

PARAMETERS

cons — When you enter the cons command without parameters, the currently selected cons parameters are displayed on the system terminal.

cyc — Specifies that the linked events may occur on any consecutive bus cycle.

fet — Specifies that the linked events must occur on consecutive fetch cycles. Note that this parameter is emulator-dependent. Refer to the Emulator Specifics section of your System Users Manual to determine if this function is available for your emulator. When fet is specified, the last event in the series must include b=f.

emu — Specifies that the linked events must occur on consecutive emulator cycles. Note that this parameter is emulator-dependent. Refer to the Emulator Specifics section of your System Users Manual to determine if this function is available for your emulator.

sequence — This parameter selects the events that form a sequence and the order of those events within that sequence. There are 12 possible event sequences, as follows:

12	123	1234
23	234	2341
34	341	3412
41	412	4123

clr Clears the currently selected cons parameters.

... This parameter indicates that more than one sequence may be entered within the same command line.

EXAMPLES

Make the occurrence of event 2, on any cycle, depend upon the occurrence of event 1, with the following command line:

> cons cyc 12 <CR>

Make the occurrence of event 3, on a fetch cycle, depend upon the occurrence of event 2 (on the preceding fetch cycle), with the following command line:

> cons fet 23 <CR>

Make the occurrence of event 2 dependent upon the occurrence of event 1, and make the occurrence of event 4 dependent upon the occurrence of event 3, with the following command line:

> cons cyc 12 34 <CR>

Display the current parameters for the cons command, with the following command line:

> cons <CR>

Clear the previously defined cons command parameters to a default condition, with the following command line:

> cons clr <CR>

NOTE

In order to use the cons command, the emulator must run at full speed. That is, the system's TRACE function and the TTA's -c option must both be disabled. In addition, the TTA stop or -s functions can be set only for the last event in a linked sequence. Note also that the TTA counter functions will only affect the last event in a linked sequence.

SYNTAX

cou all clr

cou [**-s** / **-c**] (4 / 3 / 2 / 1) (**clr** / **r**=restart / **g**=gate / **o**=output / **s**=source / **v**=value) [...]

EXPLANATION

The cou (count) command defines a counter operation. This command selects a value to be counted, a source that is counted, a gate signal that will enable or disable the counting process, and the kind of signal that will be output when the counting operation is completed.

PARAMETERS

1 Specifies that the command line applies only to counter 1.

2 Specifies that the command line applies only to counter 2.

3 Specifies that the command line applies only to counter 3.

4 Specifies that the command line applies only to counter 4.

all Specifies that the command line applies equally to all four counters.

clr Clears the current cou command programming for the indicated counter.

-s Sets a breakpoint for the trigger channel associated with the indicated event. This breakpoint interrupts the program when the trigger channel's output signal is enabled, prints a trace line, and returns control to the operating system. The -s parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-c Selects the breakpoint "continue" option. This option sets a breakpoint for the trigger channel associated with the indicated event, interrupts the program when that trigger channel's output signal is enabled, and prints a trace line. However, unlike the "stop" option, the "continue" option returns control to the program rather than to the operating system. The -c parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

value This parameter selects the value to be counted. The format is v=n. Unless specified otherwise, n is assumed to be decimal. Any valid numeric format may be used if the rules for your development system are followed. The allowable value is affected by the output parameter specified, and the gate option. See Table 2-5 for output parameter details.

source This parameter selects the signal that the counter is to count. The options for this parameter are listed in Table 2-2.

Table 2-2
Counter Source Parameters

Parameters	Signal Source for Counter N
s=200nsec	The counter's 200 nsec clock
s=2usec	The counter's 2 usec clock
s=20usec	The counter's 20 usec clock
s=200usec	The counter's 200 usec clock
s=2msec	The counter's 2 msec clock
s=ev1	The event signal for channel 1
s=ev2	The event signal for channel 2
s=ev3	The event signal for channel 3
s=ev4	The event signal for channel 2
s=trig1	The trigger signal for channel 1
s=trig2	The trigger signal for channel 2
s=trig3	The trigger signal for channel 3
s=trig4	The trigger signal for channel 4
s=acq	The INCR PTR signal (*a)
s=cyc	The SLV OPREQ signal (*b)
s=emuclk	The emulator's clock signal (*c)
s=qua	The event qualifier signal (*d)

(*a) The INCR PTR signal occurs each time the TTA stores a bus transaction.

(*b) The SLV OPREQ signal occurs on each emulator cycle.

(*c) The emulator's clock signal may be divided down in frequency before being presented to the TTA counters. Check the Emulator Specific manual for your emulator for details.

(*d) The event qualifier signal comes from the Data Acquisition Probe.

NOTE

Only one of the three counter source signals (cyc, emuclk, and qua) may be selected at one time. However, each of the four counters may operate upon the selected signal.

gate This parameter places a restriction on the indicated counter and specifies those conditions during which the counter can count. The options are listed within Table 2-3.

NOTE

The gate parameter is not allowed on Counter 1.

Table 2-3
Counter Gate Parameters

Parameters	Explanation
g=off	Removes any other counter gate restrictions
g=ctr	Counter N will count only when TCN-1 is high (*a)
g=trigh	Counter N will count only when trigger N-1 remains high
g=trigl	Counter N will count only when trigger N-1 remains low
g=seqh	Counter N will start counting when trigger N-1 pulses high
g=seql	Counter N will start counting when trigger N-1 pulses low
g=self	Counter N will count only when trigger N remains high

(*a) TCN is an internal signal that precedes the output signal circuitry for a given counter. Initially low, this signal pulses high when the counter has reached 0, then goes low again on the next counter source signal.

restart Selects the restart function. The restart function causes a counter to be reloaded with its initial "value" when the "gate" source is asserted. The options are R=on, to enable the function, and R=off, to disable the function. Note that a "gate" parameter must be selected if the restart function is used.

NOTE

The accuracy of the counters within the AM9513 is limited for certain counter operations. When a counter is incrementing, with restart on and a gate signal selected, the actual count will be 1 less than the number of source signals that occur. When a counter is decrementing, the initial value and the selection of a gate signal and the restart function will affect the accuracy of the counter, as shown in Table 2-4.

Table 2-4
Counting Restrictions

value	gate source	restart	counter output changes on
1-65535	not selected	off	source signal times value (accurate)
1	selected	off	the second source signal
2-65535	selected	off	source signal times value (accurate)
1	selected	on	the fourth source signal
2	selected	on	the fourth source signal
3	selected	on	the fourth source signal
4-65535	selected	on	source signal times value plus 1

output This parameter controls the output of the counter and also when the associated event is armed. The valid options are listed within Table 2-5. The output selected will affect the low value listed under Value Parameter in Table 2-5.

Table 2-5
Counter Output Parameters

Parameters	Explanation	Value Parameter
o=arm	The event of the selected trigger channel is armed. The counter operates independently.	V=0—65535
o=disarm	The event of the selected trigger channel is disarmed. The counter operates independently.	V=0—65535
o=pulse	The event of the selected trigger channel is armed only when the counting is complete for the interval between two successive clock sources.	V=1—65535
o=delay	The event of the selected trigger channel is disabled during counting, but is enabled after the counting is complete.	V=1—65535
o=timeout	The event of the selected trigger channel is initially armed while counting, but is disarmed after the counting is complete.	V=1—65535

... This parameter indicates that more than one of the parameters within the preceding braces {} may be entered on the same command line.

EXAMPLES

Program counter 2 to be asserted after the fourth occurrence of event 1, with the following command line:

> cou 2 v=4 s=ev1 o=delay <CR>

Program counter 3 to be asserted after the twenty-second program cycle, with the following command line:

> cou 3 v=22 s=cyc o=delay <CR>

Force the output of counter 4 high, with the following command line:

> cou 4 o=arm <CR>

Clear the counter parameters for event 3, with the following command line:

> cou 3 clr <CR>

Clear the counter parameters for all four events, with the following command line:

> cou all clr <CR>

NOTE

Three gate parameters, if selected, will affect the accuracy of certain counter operations. These gate parameters and the affected counter operations are as follows:

seql - This gate parameter causes counter n to begin counting when the trigger signal for counter n-1 pulses low. Since all four trigger signals are sent low when an emulator is halted, counter n will begin counting prematurely if the emulator is halted and restarted prior to the occurrence of the selected event. To avoid this problem, do not interrupt the execution of your program when this gate parameter is selected.

trigl - This gate parameter causes counter n to count only while the trigger signal for counter n-1 remains low. Since all four counter signals are low when an emulator is started, counter n will begin counting prematurely if a counter time-base is selected as the source and any of the pulses of this time-base occur.

trigh - This gate parameter causes counter n to count only while the trigger signal for n-1 remains high. Since the time-bases for the counters are asynchronous with reference to bus transactions, a counter operation that involves both a high frequency time-base and frequent fluctuations of the trigh gate signal may have a different result each time the program is run. To avoid this problem, limit your use of the higher frequency time-bases to counting operations that involve a small number of transitions.

SYNTAX

ctr all clr

ctr [**-s** / **-c**] { **4** / **3** / **2** / **1** } { **clr** / pattern }

EXPLANATION

The ctr (counter) command defines an event or part of an event as a specific pattern of signal states for the four counter output signals. This pattern may be defined as an event for all or any one of the four event channels.

PARAMETERS

1 Specifies that the command line applies only to event 1.

2 Specifies that the command line applies only to event 2.

3 Specifies that the command line applies only to event 3.

4 Specifies that the command line applies only to event 4.

all Specifies that the command line applies equally to all four events.

clr Clears the current ctr definition for the indicated event(s).

-s Sets a breakpoint for the trigger channel associated with the indicated event. This breakpoint interrupts the program when the trigger channel's output signal is enabled, prints a trace line, and returns control to the operating system. The -s parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-c Selects the breakpoint "continue" option. This option sets a breakpoint for the trigger channel associated with the indicated event, interrupts the program when that trigger channel's output signals is enabled, and prints a trace line. However, unlike the "stop" option, the "continue" option returns control to the program rather than to the operating system. The -c parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

pattern Selects some combination of the states of the four counter outputs to be defined as an event. The outputs of counters 1 through 4 are represented as four digits, where each digit is a 1 (high), 0 (low), or X (don't care).

EXAMPLES

Define event 1 as a counter pattern in which counters 1 and 2 output a high state and counters 3 and 4 output low states, with the following command line:

> ctr 1 1100 <CR>

Define event 2 as a counter pattern in which counters 1 and 2 output a low state and counters 3 and 4 output high states, with the following command line:

> ctr 2 0011 <CR>

Define event 3 as a counter pattern in which counters 1 and 2 output a low state and counters 3 and 4 output high states and set a breakpoint for this event, with the following command line:

> ctr -s 3 0011 <CR>

Define event 2 as a counter pattern in which counter 1 outputs a low state, counter 3 outputs a high state, and counters 2 and 4 are don't cares, with the following command line:

> ctr 2 0X1X <CR>

Clear the ctr parameters that previously defined event 3, with the following command line:

> ctr 3 clr <CR>

Clear the ctr parameters for all four events, with the following command line:

> ctr all clr <CR>

NOTE

For an application of the ctr command, see Code Timing Measurements With The TTA in Section 3, TTA Applications, of this manual.

SYNTAX

data all clr

data [-s / -c] (4 / 3 / 2 / 1) **{clr}**

data [-n] [-s / -c] (4 / 3 / 2 / 1) {data / data data}

EXPLANATION

The data command defines an event or part of an event as a specific data value or a range of data values.

PARAMETERS

1 Specifies that the command line applies only to event 1.

2 Specifies that the command line applies only to event 2.

3 Specifies that the command line applies only to event 3.

4 Specifies that the command line applies only to event 4.

all Specifies that the command line applies equally to all four events.

clr Clears the current data definition for the indicated event(s).

-s Sets a breakpoint for the trigger channel associated with the indicated event. This breakpoint interrupts the program when the trigger channel's output signal is enabled, prints a trace line, and returns control to the operating system. The -s parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-c Selects the breakpoint "continue" option. This option sets a breakpoint for the trigger channel associated with the indicated event, interrupts the program when that trigger channel's output signals is enabled, and prints a trace line. However, unlike the "stop" option, the "continue" option returns control to the program rather than to the operating system. The -c parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-n Defines the event as any data other than the values specified within the command line.

data Specifies the data value that is to be detected. Although any numeric expression that is valid for your system may be used, an absolute hexadecimal value is assumed unless noted otherwise. An "x" may be substituted for any digit within a legal numeric expression for a "don't-care" situation. Note that the number of data lines, and legal values, may vary between emulators.

EXAMPLES

Define event 1 as the data value 05, with the following command line:

> data 1 05 <CR>

Define event 2 as any data value within a range of data between 01 and FF, with the following command line:

> data 2 01 0FF <CR>

Define event 2 as any data value not within a range of data between 01 and FF, with the following command line:

> data -n 2 01 0FF <CR>

Define event 3 as a data value with "don't care" digits, with the following command line:

> data 3 0XF <CR>

Clear the data parameters previously defined as event 4, with the following command line:

> data 4 clr <CR>

Clear the data parameters previously defined for all four events, with the following command line:

> data all clr <CR>

SYNTAX

disp [**-a** | **all** | **value**]

EXPLANATION

The disp (display) command displays the contents of the TTA's acquisition memory on the system terminal. You may display all bus transactions that have been stored, or display only some number of bus transactions that you want to see. Each start/stop of the emulator is indicated by a dotted line between bus transactions in a display.

PARAMETERS

disp — When you enter the disp command without parameters, those bus transactions that have been stored since the last start/stop of the emulator are displayed on the system terminal.

value — Selects some number of those bus transactions stored in acquisition memory to be displayed. Note that the transactions are counted backward rather than forward. That is, if 10 is the selected value, the last ten transactions would be displayed, rather than the first ten transactions. Unless specified otherwise, the value is assumed to be a decimal number between 1 and 255.

-a — Displays the last 255 (decimal) bus transactions that have been stored within the TTA's acquisition memory.

EXAMPLES

Display the entire contents of acquisition memory, with the following command line:

> disp all <CR>

Display the last 12 bus transactions stored within acquisition memory, with the following command line:

> disp 12 <CR>

Display only those bus transactions that occurred during the last start/stop of the emulator, with the following command line:

> disp <CR>

Display 2-1 shows the information that is displayed on your terminal when you enter the disp command:

```
-------------------------------------------------------------------------------
 ADDR  DATA  MNEMONIC        7-PROBE-0    BUS
000100  21   LXI   H        0000 0000   M RD F
000101  00                   0000 0000   M RD
000102  05                   0000 0000   M RD
000103  06   MVI   B        0000 0000   M RD F
000104  05                   0000 0000   M RD
000105  AF   XRA   A        0000 0000   M RD F
000106  86   ADD   M        0000 0000   M RD F
000501  02                   0000 0000   M RD
00010F  00   NOP            0000 0000   M RD F

-------------------------------------------------------------------------------
```

Display 2-1

Display 2-1 illustrates the information that is displayed when you enter the disp command. This information pertains to each recorded bus transaction and includes:

ADDR — The processor address that was on the bus during a bus cycle. May contain symbolic references if debug in in use.

DATA — The data that was on the bus during a bus cycle.

MNEMONIC — If the bus cycle in question is an instruction fetch, and all transactions were acquired, disassembly of the DATA is attempted.

PROBE — Indicates the states of the eight Data Acquisition Probe lines.

BUS — Indicates what type of bus cycle occurred: instruction fetch, memory read, memory write, I/O read, I/O write, ect. Refer to the Emulator Specifics section of your System Users Manual for details.

SYNTAX

eve all clr

eve [**-c** | **-s**] { **4** | **3** | **2** | **1** } { **clr** | **q=X** | **c=**pattern | **p=**value | **b=**symbol [...] | **dn=**data [data] | **d=**data [data] | **an=**address [address] | **a=**address [address] } [...]

EXPLANATION

The eve command defines each part of an event with a single command line. That is, the eve command can be used in place of the six commands that define each part of an event. The commands replaced by the eve command are the:

- ad command
- data command
- bus command
- pro command
- ctr command
- qua command

PARAMETERS

1 Specifies that the command line applies only to event 1.

2 Specifies that the command line applies only to event 2.

3 Specifies that the command line applies only to event 3.

4 Specifies that the command line applies only to event 4.

all Specifies that the command line applies equally to all four events.

clr Clears the current definition for the indicated event(s).

-s Sets a breakpoint for the trigger channel associated with the indicated event. This breakpoint interrupts the program when the trigger channel's output signal is enabled, prints a trace line, and returns control to the operating system. The -s parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-c Selects the breakpoint "continue" option. This option sets a breakpoint for the trigger channel associated with the indicated event, interrupts the program when that trigger channel's output signals is enabled, and prints a trace line. However, unlike the "stop" option, the "continue" option returns control to the program rather than to the operating system. The -c parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

q=state Can be used in place of the qua command. The state parameter for the eve command may be any state parameter valid for the qua command.

c=pattern Can be used in place of the ctr command. The pattern parameter for the eve command may be any pattern parameter valid for the ctr command.

p=value Can be used in place of the pro command. The value parameter for the eve command may be any value parameter valid for the pro command.

b=symbol Can be used in place of the bus command. The symbol parameter for the eve command may be any symbol parameter valid for the ctr command. Any number of symbols may be used, if they are followed by a space.

dn=data [data] Can be used in place of the data command with its -n modifier. The data parameters for the eve command may be any data parameters valid for the data command.

d=data [data] Can be used in place of the data command without its -n modifier. The data parameters for the eve command may be any data parameters valid for the data command.

an=address [address] Can be used in place of the ad command with its -n modifier. The address parameters for the eve command may be any address parameters valid for the address command.

a=address [address] Can be used in place of the ad command without its -n modifier. The address parameters for the eve command may be any address parameters valid for the address command.

EXAMPLES

Define event 1 as a bus transaction during which the data FF is written to memory location FFFF, while the data AA is on the Data Acquisition Probe and is qualified by a high on the event qualifier input.

> eve 1 clr a=0FFFF d=0FF b=m wt q=1 p=0AA <CR>

Define event 2 as a bus transaction during which I/O address 5050 is written to and the four counter outputs are low, low, high, and high respectively.

> eve 2 a=5050 b=i wt c=0011 <CR>

Clear the current event definitions for event 1.

> eve 1 clr <CR>

Clear the current event definitions for all four events.

> eve all clr <CR>

SYNTAX

pro all clr

pro [**-s** / **-c**] (**4** / **3** / **2** / **1**) { **clr** / value }

EXPLANATION

The pro (probe) command defines an event or part of an event as a value that represents the states of the eight probe signals from the Data Acquisition Probe.

PARAMETERS

1 Specifies that the command line applies only to event 1.

2 Specifies that the command line applies only to event 2.

3 Specifies that the command line applies only to event 3.

4 Specifies that the command line applies only to event 4.

all Specifies that the command line applies equally to all four events.

clr Clears the current probe definition for the indicated event(s).

-s Sets a breakpoint for the trigger channel associated with the indicated event. This breakpoint interrupts the program when the trigger channel's output signal is enabled, prints a trace line, and returns control to the operating system. The -s parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-c Selects the breakpoint "continue" option. This option sets a breakpoint for the trigger channel associated with the indicated event, interrupts the program when that trigger channel's output signals is enabled, and prints a trace line. However, unlike the "stop" option, the "continue" option returns control to the program rather than to the operating system. The -c parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

value Indicates the value that is to be detected on the Data Acquisition Probe. Unless specified otherwise, this value is assumed to be binary, represented by eight digits (1s, 0s or Xs). The value may be any valid expression that evaluates to a binary number between 00000000 and 11111111. Don't-care bits (x) are allowed.

EXAMPLES

Define event 1 as probe value 05 (hexadecimal), with any of the following command lines:

> pro 1 05H <CR>

or

> pro 1 5T <CR>

or

> pro 1 000000101 <CR>

Define event 2 as a probe value with "don't care" bits, with the following command line:

> pro 2 0X111XX11 <CR>

Clear the probe parameters previously defined as event 4, with the following command line:

> pro 4 clr <CR>

Clear the probe parameters previously defined for all four events, with the following command line:

> pro all clr <CR>

SYNTAX

qua all clr

qua [**-c** / **-s**] (**4** / **3** / **2** / **1**) { **clr** / state }

EXPLANATION

The qua (qualify) command defines an event or part of an event as the state of the Event Qualifier input from the Data Acquisition Probe. The state, a high or a low, may be defined as an event for all or any one of the four event channels.

PARAMETERS

1 Specifies that the command line applies only to event 1.

2 Specifies that the command line applies only to event 2.

3 Specifies that the command line applies only to event 3.

4 Specifies that the command line applies only to event 4.

all Specifies that the command line applies equally to all four events.

clr Clears the current event qualifier definition for the indicated event(s).

-s Sets a breakpoint for the trigger channel associated with the indicated event. This breakpoint interrupts the program when the trigger channel's output signal is enabled, prints a trace line, and returns control to the operating system. The -s parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

-c Selects the breakpoint "continue" option. This option sets a breakpoint for the trigger channel associated with the indicated event, interrupts the program when that trigger channel's output signals is enabled, and prints a trace line. However, unlike the "stop" option, the "continue" option returns control to the program rather than to the operating system. The -c parameter may be placed anywhere within a command line and takes precedence over any -s, -c, stop, or cont parameter previously set for the indicated trigger channel.

state Indicates the state of the event qualifier signal that is to qualify the selected event: 1 (high) or 0 (low).

EXAMPLES

Define event 2 as a high state on the event qualifier input, with the following command line:

> qua 2 1 <CR>

Define event 3 as a low state on the event qualifier input, with the following command line:

> qua 3 0 <CR>

Clear the event qualifier parameters that previously defined event 3, with the following command line:

> qua 3 clr <CR>

Clear the event qualifier parameters for all four events, with the following command line:

> qua all clr <CR>

SYNTAX

tclr -x

tclr { **1** | **2** | **3** | **4** | **all** } **[...]**

EXPLANATION

The tclr (TTA clear) command clears the parameters previously set for the trigger channels (1-4). In addition, this command can be used to clear parameters for acquisition control (acq command parameters) and for the consecutive event function (cons command parameters).

PARAMETERS

-x Specifies that the tclr command will clear the parameters previously set for acquisition control, consecutive event functions, and for each of the four trigger channels.

1 Specifies that the tclr command clears only the parameters previously set for trigger channel 1.

2 Specifies that the tclr command clears only the parameters previously set for trigger channel 2.

3 Specifies that the tclr command clears only the parameters previously set for trigger channel 3.

4 Specifies that the tclr command clears only the parameters previously set for trigger channel 4.

all Specifies that the tclr command clears the parameters previously set for each of the four trigger channels.

EXAMPLES

Clear the parameters previously set for trigger channel 1, with the following command line:

> tclr 1 <CR>

Clear the parameters previously set for trigger channels 2 and 3, with the following command line:

> tclr 2 3 <CR>

Clear the parameters previously set for all four trigger channels, with the following command line:

> tclr all <CR>

Clear the parameters previously set for acquisition control, consecutive event functions, and all four trigger channels, with the following command line:

> tclr -x <CR>

SYNTAX

ts [**-c** / **-e**]

ts [-e] [-c] (**4** / **3** / **2** / **1**) [...]

EXPLANATION

The ts (TTA status) command causes the current status of the TTA to be displayed on the system terminal.

PARAMETERS

ts	When used without parameters, the ts command causes the complete status of the TTA to be displayed on the system terminal.
1	Specifies that only the parameters for trigger 1 will be displayed.
2	Specifies that only the parameters for trigger 2 will be displayed.
3	Specifies that only the parameters for trigger 3 will be displayed.
4	Specifies that only the parameters for trigger 4 will be displayed.
-c	Specifies that only the counter parameters of the selected trigger(s) will be displayed.
-e	Specifies that only the event parameters of the selected trigger(s) will be displayed.

EXAMPLES

When the emulator is first selected, view the status of the TTA's default parameters, with the following command line:

> ts <CR>

Display the status of the TTA parameters for trigger 4, with the following command line:

> ts 4 <CR>

Display the event parameters for trigger 4, with the following command line:

> ts -e 4 <CR>

Display the counter parameters for trigger 3, with the following command line:

> ts -c 3 <CR>

Display the event parameters for trigger channels 2 and 3, with the following command lines:

> ts -e 2 3 <CR>

Display 2-1 illustrates the TTA status display as it would appear after the following commands were entered:

> tclr -x

> eve 1 a=1000 2000 b=rd <CR>

> eve 3 an=3000 4000 b=f -s <CR>

	TRIGGER 1	TRIGGER 2	TRIGGER 3	TRIGGER 4
EVENT				
Lower Address	1000	---	3000	---
Upper Address	2000 R	---	4000 N	---
Lower Data	---	---	---	---
Upper Data	---	---	---	---
Bus	RD	---	F	---
Probe (7-0)	---	---	---	---
Qualifier (X)	---	---	---	---
Ctr (1234)	---	---	---	---
Consec CLR				
COUNTER				
Current Output	1	1	1	1
Output Mode	---	---	---	---
Initial Value	---	---	---	---
Current Value	---	---	---	---
Source	---	---	---	---
Gate	---	---	---	---
Restart	---	---	---	---
Breakpoint	---	---	STOP	---
Acquire All				

Display 2-1

Display 2-1 contains:

- current event definitions for each trigger channel (EVENT)
- currently selected cons command parameters (Consec)
- current counter programming for each trigger channel (COUNTER)
- currently selected bre command parameters (Breakpoint)
- currently selected acq command parameters (Acquire)

TS COMMAND NOTES

Upper address and data fields will be followed by one of the following letters:

R Indicates an address/data range or single address/data range or single address/data value. For example:

```
Lower Address     1000
Upper Address     1000 R
```

N Indicates an address/data exclusion (ad 1 -n 1000 2000). For example:

```
Lower Address     1000
Upper Address     2000 N
```

M Indicates the presence of don't care bits in the address/data field. M stands for Mask, a number in which a `1' value indicates a care bit, and a `0' value indicates a don't care position. The lower address/data value represents the value of each care bit. For example, 123X would be displayed as:

```
Lower Address     1230
Upper Address     FFF0 M
```

NOTE

Don't care data and address fields will be automatically expanded to show the X bits if a single trigger only is displayed (i.e.: ts 1 will do the don't care interpretation, if any).

Symbolic substitution is performed on address fields only if single triggers are displayed.

Section 3

TTA APPLICATIONS

INTRODUCTION

The Learning Guide (Section 1) provides an overview of how the TTA is used. The Command Dictionary (Section 2) describes each of the TTA commands in detail. This section contains some examples of how the TTA commands can be combined to solve typical design problems.

Essentially, this section is intended to provide you with a basic understanding of TTA applications, so that you can determine how best to combine the TTA commands to solve your own specific design problems.

The following examples are included in this section:

- Breaking on an Illegal Address
- Performance Analysis
- Asynchronous Data Transfer
- Stack Overflow
- Code Timing Measurements with The TTA
- Trigger N Arms Trigger N+1
- Pre-, Post-, And Center-Positioning of Trigger in Acquisition Memory

AN EXAMPLE PROTOTYPE

In the following examples, we'll be referring to an example prototype. The intent here is to provide you with a series of hypothetical situations that you might find while debugging your own hardware or software.

In these examples, we assume that we are working with a prototype that has the following components:

- A single microprocessor
- A single clock
- 2K of ROM, located at 0000—07FF
- 2K of RAM, located at 0800—0EFF
- A parallel to serial I/O device, located at address F001
- A read-only I/O port, located at F002
- A write-only I/O port, located at F003
- A read or write I/O port, located at F004

BREAKING ON AN ILLEGAL ADDRESS

Accessing an illegal address is a common software and hardware debugging problem. During software development, most programs are long enough to provide many opportunities to accidentally incorporate an illegal address. During hardware development, many kinds of devices, if defective or not yet correctly configured, can introduce an illegal address into your program. In either case, if your system's microprocessor tries to access an illegal address, several kinds of program faults could occur.

With the TTA, the solution to this problem is the same for both hardware and software. In our example prototype, legal addresses would be 0000—07FF (ROM), 0800—0EFF (RAM), and the four I/O ports (F001, F002, F003, F004). The following TTA commands would detect an illegal address:

> ad -n -s 1 0000 0F004 <CR>

> ad -s 2 0F00 0F000 <CR>

> g (starting address) <CR>

During program execution, these commands would cause a breakpoint, if any illegal address was accessed. The breakpoint associated with trigger 1 would occur when an address greater than F004 was accessed. The breakpoint

associated with trigger 2 would occur, when an address between 0F00 and F000 (inclusive) was accessed.

PERFORMANCE ANALYSIS

The execution time of a program is often the most critical part of software development. Although your program may function as it was intended to function, it may still need to be reworked to meet certain timing requirements.

Performance Analysis is a method of determining which parts of your program are time-intensive. With performance analysis, you can find and begin to optimize the code that will be executed most often.

In our example prototype, a program resides in ROM at locations 0000—7FFF. With the TTA, we could do a real-time performance analysis of a program that was designed for this example prototype. Let's assume that the following conditions exist:

- Our sample programs takes too much time to perform some function.
- There is a major subroutine, at locations 0100—200.
- There is a second major subroutine, at locations 0300—0600.

Since the subroutine beginning at 0300 has more code than the subroutine beginning at 0100, we could simply try to optimize this longer block first. Without performance analysis, this might be the best approach. However, if the subroutine at 0100 is executed much more often than the subroutine at 0300, the shorter subroutine could be the most logical place to start optimizing our code. The following TTA commands would tell us which subroutine is executed more often:

```
> ad 1 0100 <CR>

> cou 2 clr s=ev1 <CR>

> ad 3 0300 <CR>

> cou 4 clr s=ev3 <CR>

> g (starting address) <CR>

> ts <CR>
```

During program execution, counter 2 counts the number of times that the subroutine beginning at 0100 (event 1) is accessed and counter 4 counts the number of times that the subroutine beginning at 0300 (event 3) is accessed.

The ts command calls up the TTA status display. The results of both counting operations would be in this display.

ASYNCHRONOUS DATA TRANSFER

A microprocessor-based system cannot operate upon asynchronous data, unless that data is first processed by some kind of input routine. In addition, the input routine must complete the processing of asynchronous data within a specific time interval. That is, if the input routine does not complete the processing of one byte and acquire a second byte on time, a third byte might be "written over" that second byte.

With the TTA, the execution time of an input routine can be measured against the interval between incoming bytes of asynchronous data. To demonstrate this, we will use our example prototype. However, we must first make the following assumptions:

- F002 is the control port of an I/O device.
- A peripheral will send a 256-byte block of data to the I/O device.
- The asynchronous data from the peripheral will come at 90-usec intervals.
- The prototype's program polls control port F002.
- If data A5 is read at F002, asynchronous data is available and the input routine is called.
- The input routine, when finished, returns control to the polling program.

We would use the following TTA commands to measure the interval between each call for the input routine:

> eve 1 a=0F002 d=0A5 <CR>

> cou -s 2 v=45 s=2usec o=delay g=seqh r=on <CR>

> g (start of Program) <CR>

During program execution, trigger 1 occurs each time that data A5 is read at F002, but a breakpoint is not initiated. Counter 2 is given a value of 45 (the number it will count) and a source of 2-usec (the signal it will count). When trigger 1 occurs, counter 2 will begin to count 2-usec intervals from 45 down to 0. If trigger 1 (g=seqh) occurs before counter 2 reaches 0, the restart function agains loads 45 into the counter and starts the count over. If trigger 1 does not occur, and the counter reaches 0, a breakpoint is initiated. When a breakpoint occurs, control of the program returns to the

system. At this point, you can enter the disp command to view the information that preceded the breakpoint (in this case, the input routine).

STACK OVERFLOW

Some microprocessor-based systems require that a certain number of memory locations be reserved for temporary data storage. This "stack" area in memory is usually accessed sequentially, via an incremented or decremented address register. The use of an address register to access a memory location within the stack provides the software designer with a potential problem: if the address register is incremented too often, data intended for the stack may overflow into another area of memory.

With the TTA, stack memory can be monitored for an overflow condition during program execution. To demonstrate this, we again refer to our example prototype and make the following assumptions:

- The program exists entirely in RAM.
- Address locations 0100—01FF are designated as the stack.
- Data is loaded into the stack sequentially, beginning at 0100.
- Essential program information is stored at locations 0200—0300.

The following TTA commands would detect a stack overflow for our example prototype:

> eve 1 a=01FF b=wt <CR>

> eve -s 2 a=0200 b=wt <CR>

> cons cyc 12 <CR>

> g (starting address) <CR>

During program execution, these commands would cause a breakpoint (trigger 2), if the last location of the stack was written to on that cycle that preceded a write to the first location after the stack.

CODE TIMING MEASUREMENTS WITH THE TTA

The following discussion shows how you can use the TTA to measure the execution time of a segment of code.

Each trigger channel of the TTA was designed to offer a comprehensive set of breakpoint options, including breaking inside or outside of a specific address range. This capability, however, prevents the TTA from measuring the execution time of a segment of code with a single channel. It can, however, be measured with a combination of channels by setting up one event to start a counter and a second event to stop the counter. The counter result will display the total real time between the two events.

EXAMPLE

`A´ represents the start of the program. Let´s say you want to time code execution between points `B´ and `C´ (the first occurrence of each), and to stop execution at `D´. The following setup will produce a result in counter 3 which gives the time between event 1 and event 2 with an accuracy of plus or minus one bus cycle.

Waveforms shown for channel 1 and channel 2 are the counter outputs; waveform 3 is that of trigger 3, not counter 3.

```
>tclr 1 2 3                          A       B       C       D
                                     |       |       |       |

> eve 1  a=`address of B´ <CR>                 ________________
> cou 1 s=ev1 0=delay v=1 <CR>       _________|

> eve 2  a=`address of C´ <CR>                                 ______
> cou 2  s=ev2 o=delay v=1 <CR>      ___________________________|

> eve 3  c=10xx <CR>                           ________
> cou 3  g=self s=(timebase) <CR>    _________|        |_____
         o=arm v=0 <CR>
```

Channel 1: These commands force the output of counter 1 low until the first occurrence of event 1. Event 1 represents the start of the code segment to be timed.

Channel 2: Event 2 represents the end of the code segment to be timed.

Channel 3: This setup causes counter 3 to be gated by event 3, which is true only while the output of counter 1 is true and the output of counter 2 is false. Counter 3 must be programmed to count one of the TTA timebases, and will count only while event 3 is true. Event 3 represents the code segment under test.

The program can be stopped anytime after event `C´ occurs by an emulator breakpoint, the event 4 breakpoint, or a CTRL-C. In addition, the following two commands will cause the TTA buffer to store only those cycles (up to 255) which occurred during the counter gate:

```
eve 4 c=10xx <CR>

acq ev4 <CR>
```

Using the command file capability of the 8550 or 8560, you can even automate this process. Just create the following file (named TIMER) on your 8550 or 8560:

```
tclr 1 2 3
eve 1 a=$1
cou 1 s=ev1 o=delay v=1
eve 2 a=$2
cou 2 s=ev1 o=delay v=1
eve 3 c=10xx
cou 3 g=self o=arm v=0 s=$3
```

$1, $2, and $3 are the first, second, and third parameters of the command line, respectively. Therefore, the command

```
TIMER 100 0f3 2 usec <CR>
```

would program the TTA to measure the time from address 100H to address 0F3H in 2 microsecond units.

NOTE

Accuracy of plus or minus one cycle means that when the time from B to C is large, the measurement will be quite accurate. If the time from B to C is small, the potential one-bus-cycle error may become significant.

This technique measures the time from the FIRST occurrence of event 1 to the FIRST occurrence of event 2. You cannot make cumulative time measurements with this setup.

Counting emulator clocks (emuclk) instead of an internal time base may produce unexpected results, since emulator clock signals are often `divided down´ before going to the TTA counter chips. On the Z80, 9900/9989, 1802, 8086/88/87, and 68000 emulators, the emulator clock is divided by two. On the 8048 the clock is divided by 15 or 30, depending on the setting of clock divider jumpers on the emulator, and the ALE signal is used as emuclk. Time base counts will be accurate.

TRIGGER N ARMS TRIGGER N+1

A common situation in advanced debugging occurs when a particular event must be detected after the occurrence of a different event. The TTA can be programmed to perform `sequential´ triggering as follows.

You may wish to delay the arming of a trigger until another trigger occurs. Trigger 1 is a simple event, with no counter parameters specified. Trigger 2 is also a simple event, but you wish to enable its trigger after trigger 1 has occurred. The counter programming is:

> cou 1 clr s=ev1 v-1 0=delay <CR>

> eve 1 ... <CR>

> eve 2 ... c=1xxx <CR>

This will arm trigger 2 after the first occurrence of event 1.

PRE, POST AND CENTER POSITIONING OF TRIGGER IN ACQUISITION MEMORY

This example will show 2 ways of positioning the trigger in the acquisition memory. The first will demonstrate how to freeze acquisition without stopping the program. This method retains real-time performance but loses information after the freeze. The second method will show how to stop both the program and the acquisition, thus gaining a 255 transaction window around the trigger.

Event 4 is programmed to be the event of interest. The acquisition memory is programmed using the ACQUIRE command, to acquire the 128 acquisitions after trigger 4 occurs:

> eve 4 ... <CR>

> acq all for 128 acq aftertrig4 <CR>

The program is stopped by some means (escape, breakpoint). Acquisition will stop after 128 transactions have been stored, and the trigger will be centered in the buffer.

In the second example, event 1 is programmed to the event of interest. Counter 2 programming is:

> eve 1 ... <CR>

> cou -s 2 clr v=128 s=acq 0=delay g=segh <CR>

When trigger 1 occurs, counter 2 will begin counting. Its output will go high after 128 acquisitions have been counted. Event 2 must be cleared, and counter 1 output must be high. By changing the value field of counter 2, the trigger can be moved around in the buffer.

Section 4

TECHNICAL NOTES

INTRODUCTION

At SN B030000 and up, an improvement was made in the TTA's ability to count events, and is shown in the following example:

```
> sel 8085 <CR>

[fill memory with NOPs]

> eve 1 b=f <CR>

> cou 1 clr <CR>

> cou 2 s=evl <CR>

> eve 2 -s a=6 b=f <CR>

> g 0 <CR>

> ts <CR>   (Counter will display 7 events)
```

This improvement has resulted in a change in the way the TTA is used to count the number of times a program enters a range. See the following example.

```
> sel 8085 <CR>

[fill memory with NOPs]

> ad 1 0 0a <CR>

> cou 1 clr <CR>

> cou 2 s=evl 0=arm <CR>

> ad 3 -s 0b <CR>

> g 0 <CR>

> ts <CR>
```

In this case, the counter will not have counted the number of times the event (address range) was entered (one time), but will have counted the number of cycles executed while the event is true.

This improvement also results in a restriction on the ACQuire command. For example, when the following commands are added to the previous commands, the DISPlay will show only one acquisition:

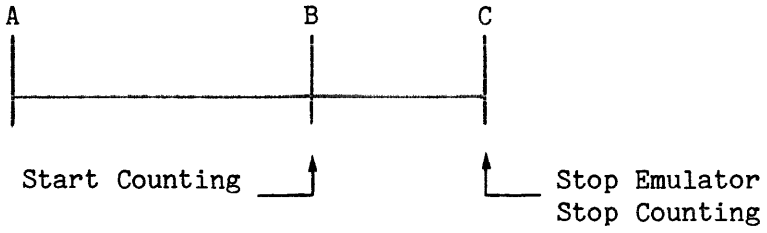
```
> acq all for 1 evl <CR>

> g 0 <CR>

> disp <CR>
```

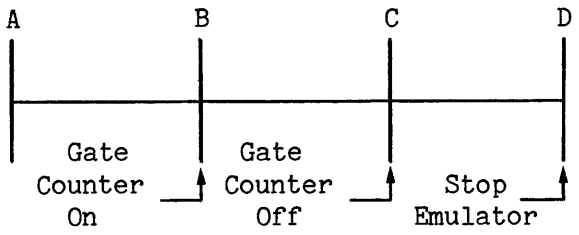
In both of the previous examples, if the source (of the cou command) is changed from evl to trigl, the TTA will operate as it did in the TTA versions B029999 and below.

One application of the counter portion of the TTA would be to time a code segment (possibly only one instruction) while stopping the emulator at the same point that the counters are stopped:



For example, assume you want to know the amount of time from B to C. In this configuration, the signal that indicates that the emulator is stopped is also used to stop the TTA counters. Due to design constraints of certain microprocessors, the emulator in question may not be able to indicate immediately that it has stopped. Therefore, if the time from B to C is very short, a large amount of error will be introduced into the timing measurements. This is particularly true for the Z8001/Z8002, 8086/8087/8088, and 68000 emulators.

The best way to accomplish this measurement is documented earlier in this section, under the heading Code Timing Measurements with the TTA. In this case, a counter is gated on at B and off at C, but the emulator is not actually stopped until sometime later, thereby eliminating the error introduced by stopping the emulator and counters at the same time. The recommended method is shown below.



DEFINITION OF SLV OPREQ(L) AND TOP SLV OPREQ(L) SIGNALS

SLV OPREQ(L) is a signal generated by the active emulator and received by the TTA. This signal is the source for all timing operations when monitoring emulator bus activity. Alternatively, TOP SLV OPREQ(L) from the top plane (used by some of the newer emulators) may be used with the same timing constraints as SLV OPREQ(L). See Fig. 4-1 for TTA timing signals.

The leading edge of SLV OPREQ(L) indicates that the emulator address and control lines are valid and is used to strobe them into the TTA latches. If the top plane is not used, the trailing edge of SLV OPREQ(L) indicates that the emulator data is valid and strobes it into the TTA data latches. The trailing edge is also used to sample the external probe inputs. If the top plane is used, data from the emulator is sampled on the leading edge of the signal TOP DATA STROBE(L), which occurs before the trailing edge of TOP SLV OPREQ(L).

Refer to the service manual for your emulator for more details on the generation of these signals and also to determine whether the top plane is used.

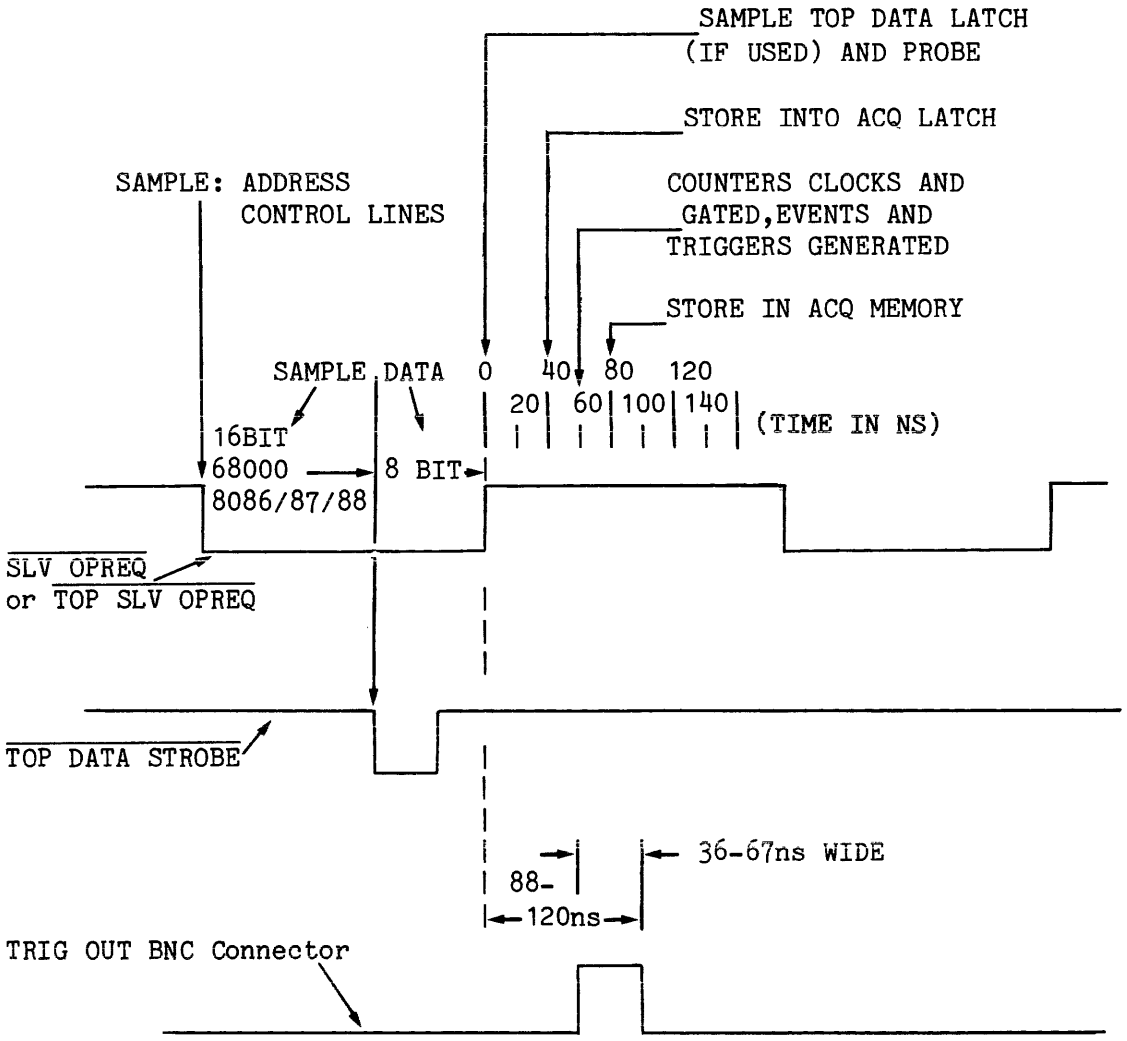


Fig. 4-1. TTA Timing Signals.

Section 5

THE DATA ACQUISITION INTERFACE

INTRODUCTION

This section describes the functions and operation of the Data Acquisition Interface and its accompanying Data Acquisition Probe. The Data Acquisition Interface is a standard accessory to the Trigger Trace Analyzer and should be installed when the TTA is installed. This section is divided into the following parts:

- The Data Acquisition Interface
- The Data Acquisition Probe
- Data Acquisition Interface Demonstration

THE DATA ACQUISITION INTERFACE

Trigger signals 1--3 (SN B030000 and up) or trigger signals 1--4 (SN B029999 and below) are output via a BNC connector on the Data Acquisition Interface. Also provided is a BNC input for the TTA's event qualifier signal, and a connector for the signal channels of the Data Acquisition Probe. Circuitry within the Data Acquisition Interface buffers and controls the signals paths to and from the TTA.

With SN B030000 and up, an Acquisition Clock Output is available via a BNC connector on the Data Acquisition Interface. This clock signal occurs when the TTA has acquired a sample of address, data, or other information. The signal is intended for operation with a TEKTRONIX DAS 9100-Series System.

The operation of the Data Acquisition Interface is controlled by its control panel and two TTA commands; pro and qua. The pro command defines 8 bits of data from the Data Acquisition Probe as an event. The qua defines the state of the event qualifier signal (1 or 0) as an event. Refer to Section 2, the Command Dictionary, for more information about these two commands.

DATA ACQUISITION INTERFACE CONTROL PANEL

The control panel of the Data Acquisition Interface provides controls and connectors for the Data Acquisition Interface's internal circuitry. Figure 5-1 illustrates the control panel of the Data Acquisition Interface. The text that follows the figure describes each connector or control.

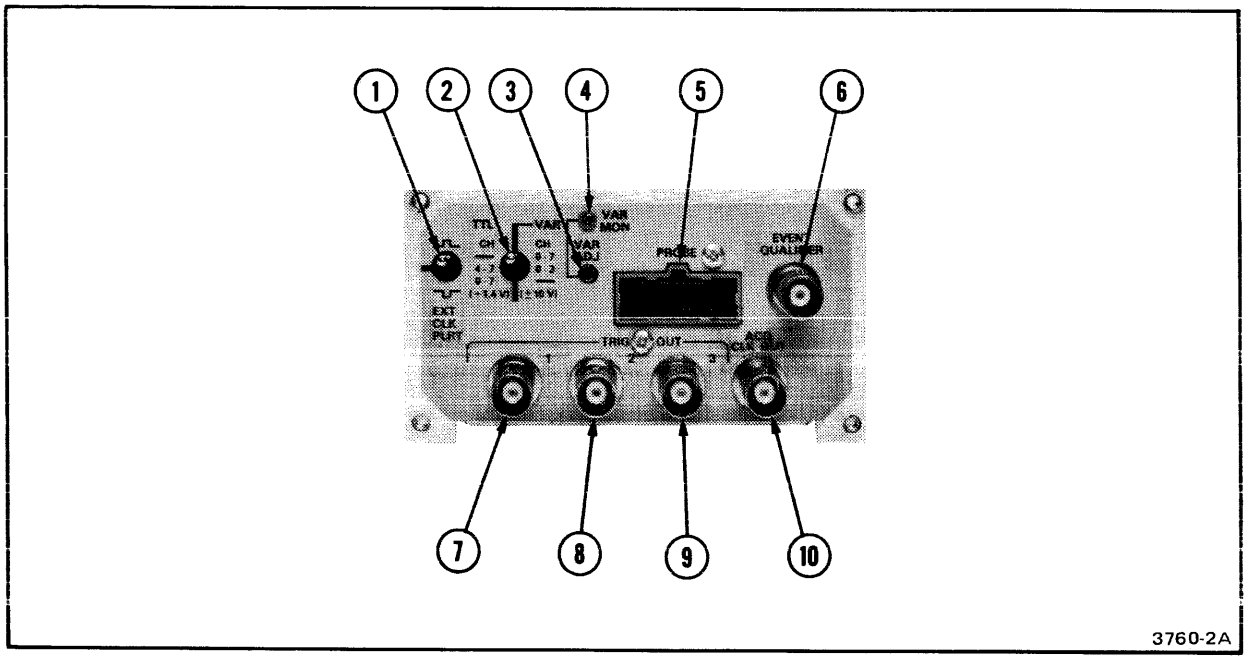


Fig. 5-1. Control Panel of the Data Acquisition Interface.

1. EXT CLK PLRT: This three-position toggle switch selects the polarity of the external clock signal. which is used to sample the eight external input signals. If the switch is up, a positive input clock must be used to sample the input signals. If the switch is centered, the signals pass transparently through the external input latch and are sampled at the end of each emulator bus cycle. If the switch is down, a negative input clock must be used to sample the input signals.

2. TTL VAR: This three-position toggle switch selects the threshold voltage level for the Data Acquisition Interface's eight data paths. If the switch is up, all eight lines may be at the variable voltage levels. If the switch is down, all eight lines can carry only TTL voltage levels. If the switch is centered, lines 0—3 are variable and lines 4—7 are at TTL levels.

3. VAR ADJ: This trimmer is used to adjust the variable voltage level selected by the TTL VAR switch.

4. VAR MON: This testpoint allows the variable voltage level to be monitored, as it is being adjusted.

5. Data Acquisition Probe Socket: This socket accepts the 25-pin connector on the end of the Data Acquisition Probe's cable.

6. EVENT QUALIFIER: This BNC connector provides a point of entry for the Event Qualifier signal that is used in conjunction with the QUA command.

7. TRIG OUT 1: This BNC connector makes the trigger signal for Event 1 available to the user.

8. TRIG OUT 2: This BNC connector makes the trigger signal for Event 2 available to the user.

9. TRIG OUT 3: This BNC connector makes the trigger signal for Event 3 available to the user.

10. ACQ CLK OUT (SN B030000 and up): This BNC connector makes the ACQ CLK signal available. The signal occurs when the TTA has acquired a sample of address, data, or other information.

    TRIG OUT 4 (SN B029999 and below): This BNC connector makes the trigger signal for Event 4 available.

THE DATA ACQUISITION PROBE

The Data Acquisition Probe (P6451) is a nine-channel active probe. Eight of the channels are used for data acquisition. The ninth channel is used for a clock signal. The probe's ten test clips are connected as follows:

- 8 leads to an 8-bit data source
- 1 lead to a clock source
- 1 lead to logic ground

The connector on the Data Acquisition Probe's cable fits into a socket on the control panel of the Data Acquisition Interface. The probe clips may be connected to external signal sources. Refer to the P6451 Data Acquisition Probe's product literature for more information.

DATA ACQUISITION INTERFACE DEMONSTRATION

This demonstration shows how to use the Data Acquisition Interface and Probe. In the example, you will connect the probe to a TEKTRONIX MicroLab I test fixture, define the data on the probe inputs as an event, and observe the effect on the TTA when this event occurs. To do this demonstration, you will need a TEKTRONIX 8500-series Microcomputer Development System, with the following options installed:

- An Emulator Processor with a Prototype Control Probe
- A Trigger Trace Analyzer with a Data Acquisition Interface and Data Acquisition Probe

In addition, you will need a TEKTRONIX MicroLab I test fixture, with a personality card installed.

GETTING STARTED

The first portion of this demonstration shows how to configure, power-up, and initialize the instruments that are used. To do this, complete the following steps:

1. With all power turned off, connect the prototype control probe to the zero-insertion-force socket on the MicroLab I's Personality Card.
2. Refer to Fig. 5-2. This figure illustrates the near edge of a circuit board that protrudes from the MicroLab I above the personality card. Along the exposed edge of this upper board is a row of jumper positions with two jumpers in place. Connect test-clip 0 (the black lead) of the Data Acquisition Probe to one of the square pins in the row of pins closest to the board edge. Note that each pin in this nearer row of square pins is tied together. Next, connect test clip 7 (the violet lead) to one of the square pins in this same row. Then, connect the ground (gnd) test clip of the Data Acquisition Probe as shown in Fig. 5-2. Do not connect the clock test clip.
3. Turn on the power to your development system.
4. Enter the sel command to select the emulator you will use.
5. Enter > em 2 <CR> to select the emulation mode.

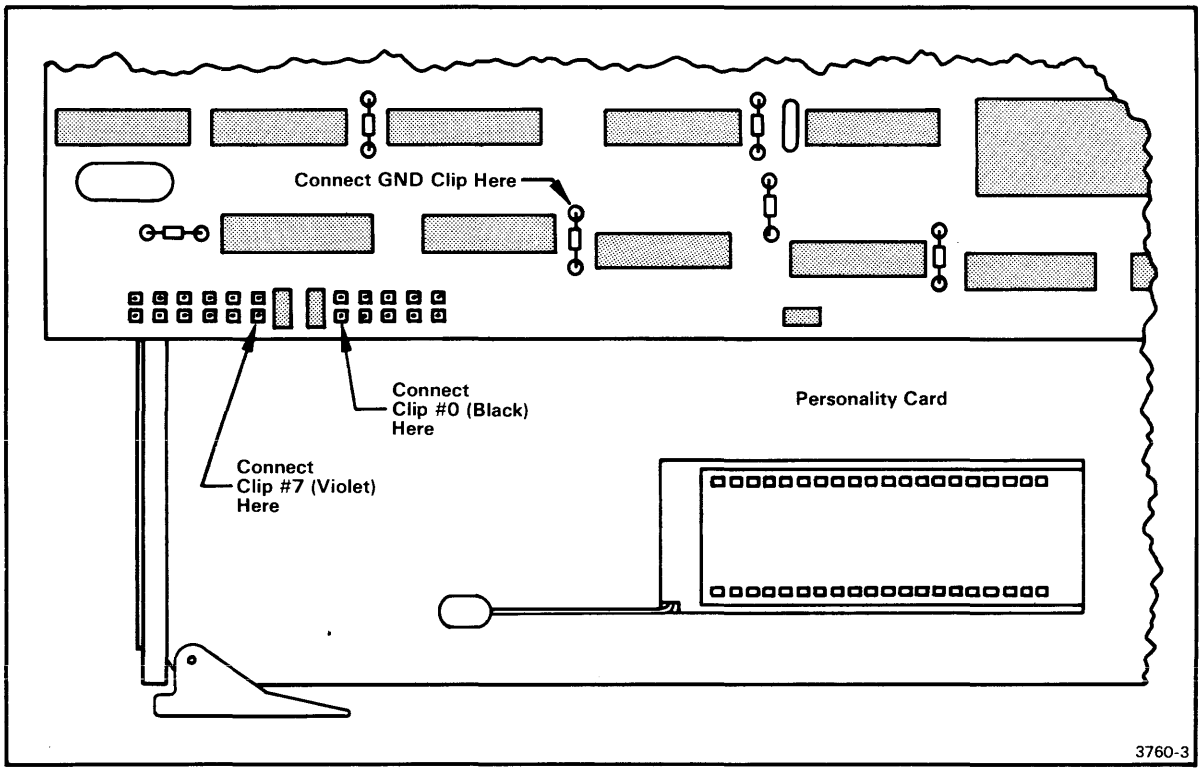


Fig. 5-2. Test-clip Positioning.

NOTE

This figure shows the positions of the test clips on the upper circuit board in the MicroLab I test fixture.

6. Turn on the MicroLab I.

7. Enter > g <CR>

8. Press the ESC key on your terminal. The system displays a trace line and returns control of the system to you.

9. Enter > disp <CR>

   Note that the display that appears contains a series of high states for the column of probe data that represents test-pins zero and seven of the Data Acquisition Probe.

10. Enter > pro 1 81H <CR>

11. Enter > cou 2 v=5 s=ev1 o=delay <CR>

12. Enter > bre 2 <CR>

13. Enter > g 00 <CR>

    A trace line should appear after control of the system returns to you. The trace line indicates that trigger 2 occurred.

14. Enter > disp <CR>

    Note that the display that appears now contains only five occurrences of event 1, which was defined by the probe command as data 81. This indicates that Trigger 2 occurred after the fifth iteration of event 1.

NOTE

The signals on the Data Acquisition Probe are sampled by the TTA at the end of an emulator's bus cycle. Since this sample interval is determined by signals internal to the TTA, there is a restriction on the use of the Data Acquisition Probe. Essentially, changes in the states of the probe signals will not be acquired by the TTA, if they precede or extend beyond the sample interval.