TEXAS INSTRUMENTS

TMS370 Family XDS/22

User's Guide

1989

8-Bit Microcontroller Family

TMS370 Family XDS/22 User's Guide

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

WARNING

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits for computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

Contents

Illustrations

Tables

Section 1

Introduction

The TMS370 debugger is a screen-oriented, interactive program that aids in the development of applications for TMS370 family microcontrollers. The debugger is used with a hardware unit called an **emulator**, which provides realtime in-circuit emulation of the TMS370 microcontroller. The debugger runs under the MS-DOS operating system on an IBM or TI personal computer (or compatible) and connects to the emulator through an RS-232 serial communications link.

The TMS370 emulator contains three boards: a communications board that connects to the PC; a breakpoint, trace, and timing board (BTT) that monitors microcontroller execution and causes hardware breakpoints; and the emulator board itself.

Attached to the emulator is a target cable with the same pinout as the device being emulated. This connector plugs directly into the application system's circuit board using the same socket that would normally hold the TMS370 microcontroller. The application system is also referred to as the **target system.**

The emulator has onboard memory that can be mapped into any part of the address space and used instead of actual target memory. This memory also simulates the operation of EEPROM and ROM on the TMS370 devices.

The TMS370 assembler and linker produce executable COFF object modules. The debugger is then used to download a program into target or emulator memory and execute it. The debugger provides interactive control of the emulator with the following features:

- Window-oriented user interface with menu-driven command structure.
- Ability to display and change registers and memory.
- Full access to symbol tables.
- A line by line patch assembler.
- A symbolic reverse assembler that displays object code.
- Full symbolic expression analysis that recognizes all assembly language operators.
- Full control of microcontroller execution, including single-step execution.
- Software breakpoints for halting execution at selected addresses.

- Hardware breakpoints and tracing, allowing detailed monitoring and execution control at the hardware level.
- Continuous run mode, allowing features listed above to be used while the microcontroller is running.
- Memory mapping that allows appropriate configuration of emulator memory.

Section 2

Getting Started

This section includes the proper procedures to set up an XDS/22 system containing an XDS/370 emulator. The following topics are covered:

2.1 Unpacking the XDS

1) Remove the XDS from its shipping carton[1] along with a cable package, then place the unit on a clean work surface as near to its permanent operating station as possible. **Be sure to lift the XDS only by grasping it from the sides and bottom. Do not use the front panel, decorating trim, or back-panel connectors as handles.**

> **Caution:**
>
> **Do not connect system to power source at this time. Wait until all installation checks are complete.**
>
> **Protect target-connector pins with the supplied plastic pin guard (or non-conductive foam) when not connected to a target system to prevent physical damage to the connector pins or electrical damage from transient voltages or electrostatic discharges.**

2) Remove the front panel of the unit as follows:

   Loosen the two thumbscrews located at the bottom of the front panel. These screws will not detach completely from the panel but will loosen completely from the main chassis. Gently pull the cover outward and down until its top edge clears the chassis.

3) Check the card-location chart inside the card-cage cover against the actual cards installed.

4) Remove shipping retainer from XDS/22 interior.

5) Reseat any card that appears loose in its socket. Do this by pulling out on the inside edge of the card ejectors then firmly pressing the card back into its socket by pressing in on the card ejectors. It may be necessary to move the card slightly up or down within its slot.

> **Caution:**
>
> **To avoid damage to the card or injury to yourself, do NOT grasp the top or bottom of the card.**

6) Make sure all required cables are connected and secured.

7) Replace the front panel and tighten the screws. Be careful not to snag or crimp the cables exiting from the card cage.

---

[1] Save the carton for possible reuse if the unit must be shipped.

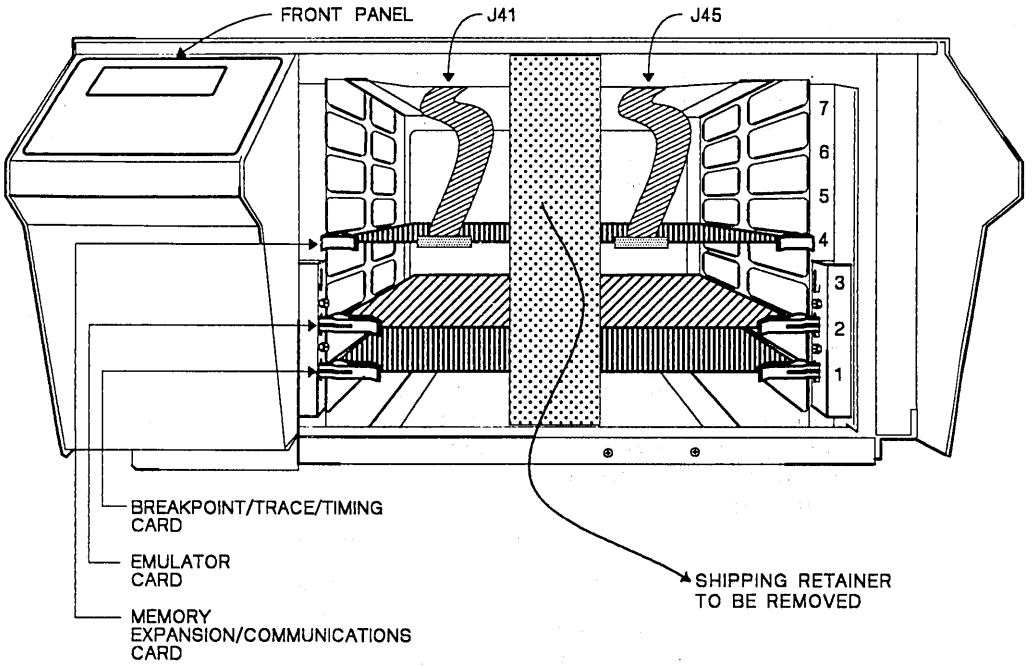


Figure 2-1. XDS/22 Interior as Shipped

2.2 Kit Contents

Please verify that you received the components listed in the following table by matching the part number given with the number printed on each component.

Table 2-1. Contents of TMS370 XDS Kit

Part Number	Description
2310990-1	XDS/22 chassis (USA)
2243660-2	Breakpoint/Trace/Timing board
2536810-1	TMS370 emulator board
2539628-1	Target cable assembly (68-pin PLCC)
2539634-1	Target cable assembly (28-pin PLCC)
2311050-2	Memory Expansion/Communications board

The following are also included in the XDS shipping container.

- XDS/22 chassis with 370 emulator, communications, and BTT boards
- Interface cables
  - EIA 25-pin cable (male-male/female)
  - 25-to-9 pin adaptor
- Power Cord
  - USA cord terminated for 5-15R wall socket, **or,**
  - European cord unterminated unless country was specified by customer at order time
- BTT Probes
- 68-pin PLCC cable
  - 68-pin PLCC-PGA adapter (installed)
  - 68-pin PLCC surface mount socket
- 28-pin PLCC cable
  - 28-pin PLCC-PGA adapter
  - 28-pin PLCC-DIP adapter (installed)
  - 28-pin surface mount socket
- 2 grounding clips with E2 hook terminations
- Documentation kit

2.3 Environmental Requirements

A minimum clearance of 5 inches (12.7 cm) must be maintained between the XDS unit and surrounding walls and/or equipment to permit sufficient air flow.

Temperature and humidity requirements, for both operation and storage, are given in the following table:

Table 2-2. XDS Temperature and Humidity Ranges

Operation		Storage	
Temperature	**Humidity**	**Temperature**	**Humidity**
60°F - 90°F 16°C - 32°C	30% - 80%	-40°F - +185°F -40°C - +85°C	5% - 90%

2.4 Emulator/Debugger Cabling

The debugger host machine connects to the emulator with a serial RS-232 communications link. This section briefly describes the interface so that any problems can be corrected.

Three standard RS-232 signals plus ground are used to implement the communications protocol. Two data lines transmit characters back and forth. An "attention" line from the emulator is used to alert the debugger when events such as breakpoints are encountered.

The pin-out at the host and emulator for these signals is illustrated below. The host pin-out shown is for a standard DB-25 connector for the host serial port. The IBM AT serial port has a DB-9 connector instead of the standard DB-25, and its pin-out is shown in parentheses.

The EIA cable supplied with the XDS unit is wired as shown below, and must be connected between Port A of the XDS and the serial port on the personal computer. No additional cabling or handshaking is needed.

Table 2-3. Emulator - Host Cable RS-232 Pin-outs

	EMULATOR (male DB-25)			HOST (DB-25)		(DB-9)
Function	Pin	Signal		Signal	Pin	(AT)
Connection established	8	DCD	<----------	DCD	8	(1)
Data to host	3	TX	---------->	RX	3	(2)
Data to emulator	2	RX	<----------	TX	2	(3)
Terminal ready	20	DTR	<----------	DTR	20	(4)
Signal ground	7	GND	----------	GND	7	(5)
Emulator ready	6	DSR	---------->	DSR	6	(6)
Attention to emulator	4	CTS	<----------	RTS	4	(7)
Attention to host	5	RTS	---------->	CTS	5	(8)
Not used	22	RI	----------	RI	22	(9)

2.5 AC Input Requirements

Ensure the compatibility of the local power source with the XDS/22 power requirement (Table 2-4) shown on the back-panel model plate (Figure 2-2). Make sure the POWER switch is in the OFF position.

Do not connect any electric motors or fluorescent lights to the XDS/22 power circuit. Noise and/or voltage spikes from these devices could affect operation.

Table 2-4. XDS/22 Chassis Power Requirements

Rated Voltage	Rated Frequency	Rated Current	Wall Outlet Specification
100 V_{ac}	50/60 Hz	3 Amps	5-15R
115 V_{ac}	60 Hz	3 Amps	5-15R
220 V_{ac}	50 Hz	2 Amps	Schuko 1050
240 V_{ac}	50 Hz	2 Amps	BSI 1363

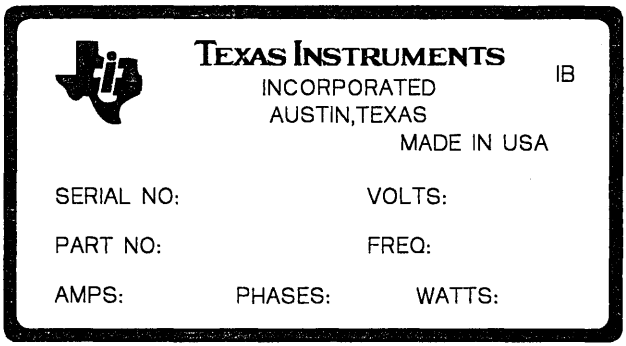


Figure 2-2. XDS/22 Model Plate

2.6 Software Installation

This section contains step-by-step instructions for installing the TMS370 assembly language tools package. This package can be installed on the IBM PC (running PC-DOS[2]) and the TI PC (running MS-DOS[3]).

The TMS370 software package is shipped on one double-sided, dual-density diskette. There are two distributed versions, one for the TI-PC and one for the IBM and IBM-compatible PCs. You need the following items to use the package:

- IBM-PC or IBM-compatible PC, TI-PC, or MS-DOS compatible PC
- 512K or more RAM memory
- Serial communications port
- Floppy disk drive (to read the TMS370 software disk)

In addition, the following are recommended:

- Fixed disk drive
- Color monitor

The following instructions are for both fixed disk drive systems and dual floppy drive systems. On a dual-drive system, the PC/MS-DOS system diskette should be in drive B. The instructions use these symbols for drive names:

A: Floppy disk drive for hard disk systems *or* source drive for dual-drive systems.
B: Destination or system disk drive for dual-drive systems.
C: Winchester (hard disk) for hard disk systems. (**E:** on TI PCs.)

1) Make backups of the product diskettes. First format a blank diskette. Insert a blank (destination) diskette in drive A. Enter:

```
FORMAT A: <CR>
```

When PC/MS-DOS prompts: `FORMAT ANOTHER (Y/N)?`, respond with `N`. Now copy the disks.

On *hard disk* systems, enter:

```
DISKCOPY A:  A: <CR>
```

Follow the system prompts, removing and inserting the product and blank diskettes as directed. When PC/MS-DOS prompts: `COPY ANOTHER (Y/N)?`, respond with `N`.

On *dual-drive* systems, place a product diskette in drive A: and a blank, formatted diskette in drive B. Enter:

```
COPY A:*.*  B:*.* <CR>
```

---

2 PC-DOS is a trademark of International Business Machines.

3 MS is a trademark of Microsoft Corporation.

2) Create a directory to contain the TMS370 software.

   On *hard disk* systems, enter: `MD C:\370TOOLS <CR>`

   On *dual-drive* systems, enter: `MD B:\370TOOLS <CR>`

3) Copy the TMS370 tools onto the hard disk or the system disk.

   On *hard disk* systems, enter: `COPY A:\*.* C:\370TOOLS\*.* <CR>`

   On *dual-drive* systems, enter: `COPY A:\*.* B:\370TOOLS\*.* <CR>`

You will need to make two changes to the system files on your PC. First, the `CONFIG.SYS` file must contain the line:

```
FILES=20
```

Second, you must add a string to the DOS environment to tell the debugger where display configuration and help files are stored. Use the DOS command `SET` to define the environment variable `IPCDIR` as follows:

```
C> SET IPCDIR=E:\370TOOLS
```

If the symbol `IPCDIR` is not found in the environment, the debugger will look in the current directory for configuration or help files. You can put the `SET` command in your `AUTOEXEC.BAT` file to define the symbol each time the system is booted. (This is also discussed in Section 9.3, Debugger Color Setup.)

You do not have to be in your debugger subdirectory to run the debugger; the DOS `PATH` command will allow DOS to find the executable debugger file. Refer to your DOS manual for examples of how to use the `PATH` command.

2.7 Power-up Procedure

> **Caution:**
>
> **Always ensure the power to the XDS is off before connecting or disconnecting connectors or installing or removing circuit boards.**
>
> **Remove and install circuit boards and connectors only in a static-free workstation following normal MOS-device static-handling procedures.**

If a target system is to be used, replace the TMS370 microprocessor in the target system with the target connector of the XDS. See Section 11.5 for connection to a target system.

> **Caution:**
>
> **When replacing the TMS370 microprocessor in the target system with the target connector, make sure pin orientation is correct. Pin 1 is located on the side opposite the cable.**

Connect the power source to the unit by plugging the power cord into the wall outlet.

Apply power to the unit by pressing the POWER switch.

Verify that the POWER light is on. If not, turn power off immediately and perform the following checks:

1) Check that power is available from the source.

2) If power is present but the unit is not operating, unplug the unit and check the fuses.

2.8 Invoking the Debugger

Before invoking the debugger, be sure that the emulator is plugged in and that the cable from the emulator to the PC host is properly connected. The debugger will not operate unless the emulator is connected and turned on.

The command to run the debugger from MS-DOS is:

DBR370 [/b=<baud>] [/p=<port>] [<file1> <file2> ...]

Example: DBR370 /b=2400 /p=2 test.out test.mmp

Options:

<baud> The default baud rate is 19200. If for some reason 19200 baud cannot be used, specify a new baud rate with this option. Valid baud rates are 1200, 2400, 4800, 9600, and 19200.

<port> A number from 1 to 4 that specifies which serial port on the host PC is connected to the emulator. If no port is specified, port 1 is used.

<files> Optional arguments that specify one or more files to be loaded into the debugger.

> **Note:**
>
> The command for the TI-PC version is DBR370T. Make sure you have the correct version for your system.

After the debugger is invoked, it clears the screen and displays a version number and copyright message and prints `Initializing ...` on the bottom line of the screen while it resets the emulator. After a short delay of 3-10 seconds, the top-level screen is displayed. If a filename was specified on the command line, the debugger immediately loads the file. See Loading a File – The L Command, page 5-20 for a description of how files are loaded. If the screen shows an error, press the white XDS reset button and retry.

2.9 Communications Problems

The emulator and debugger communicate at 19200 baud, with eight data bits and one parity bit. The emulator automatically matches baud rate when it is initialized by the debugger.

Port 1 is used as the default serial port if the host PC has multiple serial ports. A different port can be used for the emulator via the "/p=<port>" option on the command line, where <port> is a number from 1 to 4 that specifies the port number.

If the debugger tries to perform an operation and the emulator either does not respond within a five second time-out period or responds incorrectly, the debugger prints an error message on the bottom of the screen:

```
"Emulator not responding correctly:  Abort, Retry?"
```

This message indicates that one of several problems may have occurred, including:

- The emulator is not turned on.
- The host communications port cannot be initialized.
- The cable between the emulator and the PC is not properly connected.
- The TMS370 device is in an improper state, causing the emulator to "hang".

If you respond to the message by typing **A** (Abort), the debugger immediately exits and returns to the operating system. Typing **R** (Retry) causes the debugger to try to reset the emulator. If successful, the operation will be tried again. If not, the message will be re-displayed. If the debugger fails after several attempts to reset the emulator, you should manually reset it by pressing the reset button on the XDS system. (See Section 11.1.1, Operator Panel.)

Section 3

Functional Overview

This section summarizes the main features of the TMS370 debugger. The following topics are included:

3.1 Window-Oriented User Interface

Debugging a system requires attention to a number of different areas: the code being executed, the registers of the target machine, the variables in the program, etc. The debugger models this by providing a set of **windows** on the screen. Each window contains information pertinent to one aspect of the debugging process. You can move from window to window to perform specific operations such as moving to the code window to examine code or moving to the CPU register window to clear a register.

In addition to windows, the top-level debugging screen consists of the following elements:

- Top line: menu of single letter commands
- Second line: display of current system status
- Bottom line: active function key assignments

An example of the top-level screen is shown in Figure 3-1.

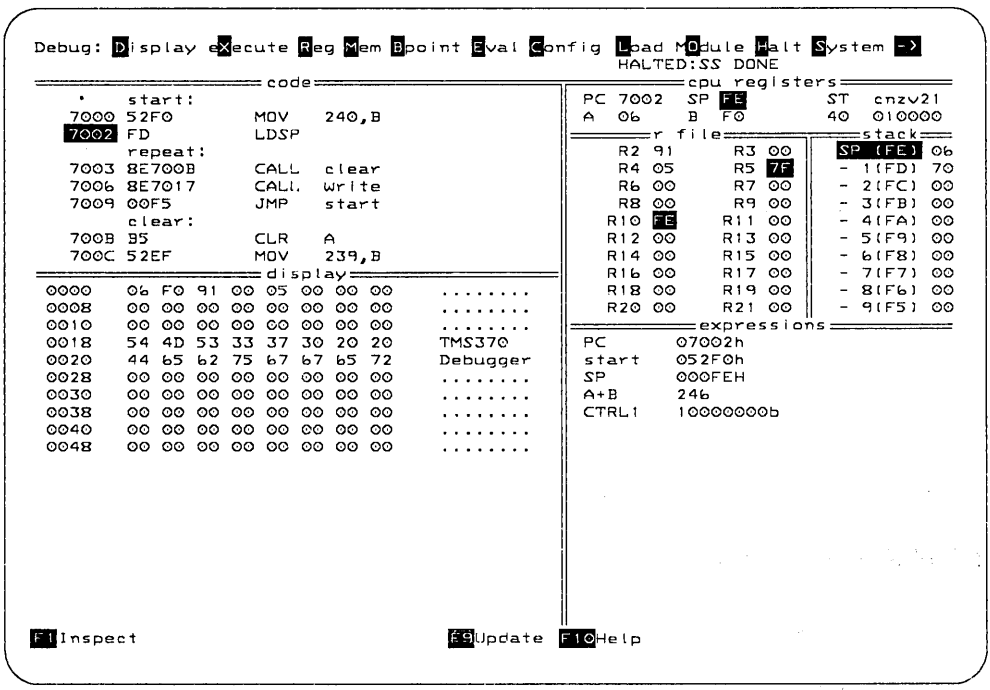


Figure 3-1. Top-Level Screen

Some windows are fixed; that is, they always monitor the same thing. Others are dynamic: you can tell the debugger what to display in the window.

Windows are based on **virtual** buffers in the debugger. This means that the debugger keeps track of more information than is actually displayed in a given window. This allows the window to be scrolled up and down in a local area without having to go to the emulator to get new data for the buffer. The window can, of course, move outside the local area, in which case the debugger automatically updates the buffer with new data from the emulator.

Windows on the screen are automatically updated whenever the microcontroller has stopped running. You may also specifically request that the screen be updated at any time by pressing the *Update* function key. When a window containing registers, memory or expressions is updated and a value has changed, the new value is highlighted so that changes can be easily spotted on the screen. See Section 4.7, Updating the Screen.

As shown in Figure 3-1, the top level screen is divided into six windows. Except for the **display window** in the lower left corner of the screen, each window is dedicated to displaying a specific type of information. The six top level windows are described in the following sections.

3.1.1 Code Window

The code window is located in the upper left corner of the screen and is dedicated to displaying the code being debugged. The code in this window is disassembled from object code in memory.

The disassembler uses the symbol table and inserts labels into the disassembly wherever possible, with the exception of relocatable register symbols. The instruction at the current PC is identified with a highlighted address. Also, instructions at which simple breakpoints have been set are marked with a breakpoint number to the left of the address. Immediate values in instructions are displayed in hexadecimal.

You can scroll down through the code, adding or removing simple breakpoints. Scrolling back up through the code works normally up to the top of the debugger's virtual buffer. Scrolling above the top of the virtual buffer is allowed one byte at a time. This can cause the entire code window to be disassembled incorrectly because the disassembly can start in the middle of an instruction. The disassembly will correct itself as the user continues to scroll upward in the window. The new location to which you scroll will become the new top of the virtual buffer.

3.1.2 Display Window

The display window is located in the lower left corner of the screen. The window displays one of several miscellaneous debugger outputs such as:

- Memory, dumped in hexadecimal and ASCII formats
- Peripheral file registers
- Symbols in the symbol table
- Module names
- A file

The functions (Fkeys) available depend on what is being displayed.

3.1.3 CPU Registers Window

Five TMS370 registers (A, B, PC, SP, and ST) are displayed in the CPU registers window in the upper right corner of the screen. This window cannot be scrolled. You can move to this window and modify the contents of any of these registers.

3.1.4 Register File Window

The register file window is located on the left, under the CPU registers. The window displays the contents of 20 registers from the register file. This window can be scrolled up and down to see different areas of the register file, as well as change the contents of any register shown.

3.1.5 Stack Window

The stack window is located to the right of the register file window and contains the contents of the current program stack within the register file. The stack window differs from the register file window in that a) when updated, the stack window automatically changes the display to reflect the offset of each register from the current top of stack, and b) the registers are displayed in reverse order, so that "higher" on the stack (i.e., closer to the top of stack) corresponds to higher in the window.

3.1.6 Expression Window

The expression window is located in the lower right corner of the screen. This window is used to display expressions specified by the user.

3.2 Command Language

The debugger command language allows you to control the debugging process. The top level of the command language is a menu of single-letter commands. Each command is invoked by typing the letter, upper or lower case, for the command. Many of the commands require additional input and qualification, so they either prompt you for input or display a menu of subcommands.

The command language is designed to be both simple for the inexperienced user and efficient for the expert. This is accomplished by limiting command menus to just one or two levels, so that nearly everything in the debugger can be controlled by a simple two-letter command with no wasted keystrokes.

The ten function keys complement the command language by providing additional control of the debugger itself. Function keys are used to scroll windows up and down, move between windows, etc. The actual function keys displayed and available at any one time are activity-dependent.

The escape key is used throughout the debugger to exit from the current operation and return to the previous command level. In this manual, the escape key is referred to with the symbol **<Esc>**. Similarly, the "Return" or "Enter" key is designated as **<CR>**.

3.3 Object Modules

The debugger can execute COFF (Common Object File Format) executable linked object modules. When the debugger loads an object module, instructions and data are downloaded into the address space of the target system (into either target or emulator memory, depending on the current configuration of the memory map). The complete symbol table from the file is also loaded into internal debugger memory.

The object module's symbol table contains information about each symbol in the program, including the symbol name, storage class, type, size, and value. Upper and lower case are significant in symbol names. The storage class of a symbol indicates whether the symbol is *global* (visible to other modules) or *static* (visible only in the module in which it is defined).

The object module's symbol table also contains the names of the source modules and identifies which static symbols were defined in which modules. The debugger uses the concept of a **current module** to identify static symbols. Only the static symbols in the current module can be accessed at any time. To access statics in other modules, the current module must be changed. Global symbols are always visible to the debugger.

The symbol table is used in two ways: the expression analyzer uses it to evaluate symbols in expressions and the disassembler uses it to insert symbols into the disassembly. The disassembler does not guarantee that symbols show up in the disassembly whenever they were used in the source; instead, the disassembler tries to find symbols that match labels and 16-bit operands whenever it can. Eight-bit *(.reg)* symbols are not used by the disassembler. Symbols defined in the symbol table may be used by the line by line assembler, but new symbols can not be defined by the line by line assembler.

3.4 Expressions

Whenever the debugger prompts for an address or a value, you may enter an arbitrarily complex expression. The debugger evaluates expressions using the symbol table and the emulator. Expressions may consist of numeric constants, symbols, and register names, separated by operators. Any expression used in the assembler source will evaluate to the same value when used in the debugger. Spaces are insignificant in expressions.

Expressions are evaluated using 16-bit signed arithmetic. Both intermediate and final vaiues for expressions can therefore range only from -32768 to +32767. If a value exceeds this range, the displayed result will be incorrect.

You will find more information on expression evaluation in Section 5.6, Evaluate an Expression - The E Command on page 5-11.

3.4.1 Constants, Symbols, and Registers

Numeric constants can be hexadecimal, octal, binary, or decimal. Constants are identified with a suffix, as follows:

Radix	Suffix
hexadecimal	H or h
octal	Q or q
binary	B or b
decimal	D or d

The first character of a hex constant must be a decimal digit to distinguish it from a symbol name. For example, *Fh* is an invalid constant; *0Fh* is the correct form. Numbers that have no suffix and do not begin with a zero are assumed to be decimal.

Symbols are identifiers of up to 32 characters in length as stored in the COFF symbol table. Case is significant in symbol names. The valid characters are a...z, A...Z, 0...9, $, _. Symbols must not begin with a number.

Register names are identifiers for the standard TMS370 registers. Upper or lower case may be used for all register names. Register names are:

PC	Program counter
SP	Stack pointer
ST	Status register
A	Register A (R00)
B	Register B (R01)
Rn	Register n †
Pn	Peripheral n †

† **n** is a number from 0 to 255 (decimal)
or 00 to 0FF (hex)

3.4.2 Operators

Table 3-1 lists the operators that can be used in expressions. Operators are listed in order of highest to lowest precedence. Operators with higher precedence are applied first during expression evaluation. Operators in the same section of the table have the same precedence. Parentheses can be used to alter the order of evaluation.

Unary operators are used by prefixing the operand with the operator (*-x , @sym*) and are right associative (*@-a = @(-a)*). Binary operators are used by infixing the operator between the operands (*a+b , x mod y*) and are left associative (*a+b-c = (a+b)-c*). Logical operators interpret any non-zero operand as TRUE; the result of logical operators is either 1 for true or 0 for false.

Table 3-1. Valid Operators

Operator	Function
+x	Unary positive; no effect
-x	Unary negative; twos complement
~x	(COM) Ones complement
!x	(NOT) Logical negate; returns 1 or 0
HI X	High byte of x
LO X	Low byte of x
*x	Contents of location x
@x	Indirection; value pointed to by the contents at memory location [x-1]:[x]
#x	Address of; use 16-bit address of symbol x
x * y	Multiplication
x / y	Integer division; quotient truncated
x % y	(MOD) Modulo
x << y	(SHL) X shifted left by y bits
x >> y	(SHR) X shifted right by y bits
x + y	Addition
x - y	Subtraction
x & y	(AND) Bit-wise AND operation
x | y	(OR) Bit-wise OR
x ^ y	(XOR) Bit-wise exclusive OR
x < y	True if x less than y
x <= y	True if x less than or equal to y
x > y	True if x greater than y
x >= y	True if x greater than or equal to y
x == y	True if x equal to y
x != y	True if x not equal to y
x && y	True if x and y are both true; logical AND
x || y	True if x or y is true; logical OR

Note: Operators in parentheses indicate alternate form.

3.5 Displayed Values

Numeric values are displayed on the screen in one of two ways, in standard suffix numeric form as described above, or in *implied hex* notation. When you cannot change the radix of displayed values, as in the register or peripheral file windows, values are displayed in hex with no suffix. Anywhere you have the option of selecting different radices for displayed values, as in the expression window, values are displayed with a suffix to indicate the selected radix. Decimal values are displayed with no suffix.

When you have the option of changing the format, the debugger uses a submenu of radix selections from which you select the desired format. The options and corresponding formats are described here:

Option	Format
X	Hexadecimal
D	Decimal
O	Octal
B	Binary
A	ASCII †

† Non-printable characters displayed as '.'

3.6 Saving and Restoring Configurations

The debugging environment includes a number of configurations and set-ups that could be tedious to recreate each time the debugger is invoked. To avoid this, the debugger can save and restore a number of these configurations using disk files. In addition, file load operations can be performed under the control of a single command file, so that many set-up files can be loaded with one operation. The configurations that can be saved and restored in this way are:

- BTT settings
- Simple breakpoint addresses
- Device configuration
- Expressions in the expression window

3.7 Error Messages

If you try to perform an action that is illegal, the debugger displays an error message on the bottom line of the screen and prompts you to "hit esc key". Press the "esc" key, the message is erased, the function key help line reappears, and the debugger continues its normal operation.

Under certain abnormal circumstances, the debugger cannot communicate with the emulator. If this occurs, the debugger displays the message

```
Emulator not responding correctly - Abort, Retry, Ignore?
```

The reasons for this message and possible solutions are described in Section 2.9, Communications Problems, and Section 11.8, XDS/22 System Repair Guide.

3.8 Online Help

The debugger has a built-in interactive help system that can be invoked from anywhere in the debugger by pressing the F10 key. The debugger opens a temporary window on the screen and displays help information. Help is context sensitive; the text displayed depends on what operation is in progress when F10 is pressed. From some help screens, additional levels of help can be invoked to explain the current operation in more detail.

3.9 Execution Modes

There are two different run modes in the emulator: **normal** and **continuous**. Each mode has different ways of actually running a program.

In **normal run mode**, the CPU and BTT run together as a unit and cannot be interrupted until one halts, in which case the other is halted also. Execution can be halted by the user, by the BTT, by a simple breakpoint, or by some exception condition such as an illegal opcode or a memory access violation. While running, the emulator cannot perform any operations except checking the status of the CPU or forcing it to halt. Thus, the CPU must be halted before the debugger can see any of the effects caused by running.

In **continuous run mode**, like normal run mode, the CPU and BTT start as a unit. No operations can be performed while both are running, and if the CPU halts, the BTT halts also. However, the BTT can be halted while the CPU continues to run. This allows the emulator to perform nearly all operations that are normally possible. This includes dumping the contents of the trace buffer, reconfiguring the BTT, updating the screen, and modifying the contents of registers and memory. Whenever the debugger accesses registers and memory, the CPU is halted just long enough to perform the operation and is then restarted. Thus, continuous run mode is not truly realtime unless no memory or register accesses are requested.

The BTT can be restarted in continuous run mode without interrupting the CPU. Single-step execution is not possible in continuous run mode.

3.10 Breakpoint, Trace and Timing (BTT) Board

The Breakpoint, Trace and Timing (BTT) board in the emulator provides the capability to monitor system operation at a hardware level. The board can be programmed to take various **actions** that are triggered by the occurrence of a specified combination of qualifiers. These qualifiers may include a particular address or range of addresses on the address bus, a particular data value or range of data values on the data bus, the type of memory cycle, and signals from up to eight external logic probes that can be connected to the application system.

The board has three timers, two of which can be started or stopped by qualifying an event and one other that is free-running. The timers allow timing statistics to be taken, such as percent of time spent in a certain region of code.

There is a 2K x 104 bit trace memory in which all the qualifiers described above plus a time stamp from the free-running timer can be stored for later inspection.

The BTT is always in one of four logical **states**. Up to four actions can be qualified per state, with certain restrictions. A qualified event can cause a transition to another state with another set of qualifiers and actions. In this way, multi-level or sequenced breakpoints can be applied to solve complex debugging problems.

The actions that can be taken on the basis of qualified events are:

- Halt the processor (breakpoint)
- Halt the BTT (continuous run mode only)
- Store the qualifiers in a trace sample memory
- Start or stop the timers
- Cause a transition to a different state

3.11 Memory Mapping

The emulator board contains 64K bytes of RAM reserved for emulation. Memory references from the TMS370 can be satisfied in one of three ways:

- Internally from the TMS370 device,
- Externally from target system memory, **or**
- Externally from emulator RAM.

The 64K byte address range of the microcontroller is divided into 16-byte areas called **frames**. The memory mapping scheme of the debugger and emulator allows you to specify, for each sixteen-byte frame of the address space, where memory references to that frame are to be satisfied. References to the register file (addresses 0–0FFh) are always satisfied internally. Sixteen-byte frames in the peripheral register file (addresses 1000h–10FFh) can be mapped either internally or to an optional user-supplied peripheral expansion board on the emulator.

In addition to mapping various parts of the address space to different parts of the physical memory, you can protect areas of memory from certain types of access. Protection can be on read cycles, write cycles, both, or none. If the CPU tries to execute a memory cycle on an address that is protected for that cycle, a trap occurs and the CPU is halted. Thus, memory can be used as ROM by write protecting it, and it can be treated as non-existent by both read and write protecting it. Any memory not explicitly mapped by the user is fully protected in this way so that any access to an address outside configured memory, as defined by the user, causes a trap and halts the CPU.

Some TMS370 devices contain regions of built-in EEPROM that can be programmed by the CPU at run time. The emulator has no internal ROM or EEPROM. Instead, areas of emulator RAM can be mapped in to emulate ROM and EEPROM. This allows the emulator hardware to support non-real time simulation of EEPROM programming.

The emulator traps accesses to the EEPROM control register and EEPROM memory addresses, detects what operation is being performed and simulates the behavior of the actual EEPROM programming using emulator RAM. The simulation is not completely accurate, and it is not done in real time. The CPU is halted when the trap occurs and restarted after the operation is completed, resulting in a delay of approximately 12 ms.

Emulation of ROM is performed simply by write-protecting the ROM area of emulator memory.

Section 4

Operation

This section includes the following topics:

4.1 Command Menus

Overall operation of the debugger is through command menus. Command menus are lists of command names, each of which is displayed with one highlighted letter, usually the first character in the name. The highlighted letters are the keys used to invoke the corresponding commands. Some menus have a default command letter displayed in square brackets after the list of commands; the default can be selected by simply pressing <CR>. The top level menu has more commands than can fit across the width of the screen. Use the left and right arrow keys to display the additional commands.

Command letters can be in upper or lower case. Any character that is not either a command letter in the menu or some control key (such as an arrow, function key, or <Esc>) is ignored. When a valid command letter is selected, the debugger clears the command line, displays the name of the selected command, and executes the command. Most commands require additional information, in which case the debugger either prompts for parameters or displays a submenu of commands.

4.2 Function Keys

Function keys F1–F10 are used to invoke various actions in the debugger in addition to the command menus. There are four disjoint sets of function keys: one set for the top level command menu, one set for inspect mode, one set for programming the BTT, and one set for setting up the memory map and configuration.

Each key has one unique function within a function key set. For example, in inspect mode, F3 always scrolls a window down and F4 always scrolls it up.

Some keys have the same function throughout the debugger. For example, F1 and F2 are used to move forward and backward between windows, and F10 is always the help key.

Only a subset of the keys within each set may be valid at any given time. Currently valid keys are displayed on the bottom line of the screen for reference.

4.3 Inspect Mode

The F1 key is used from the top level command line to move into the windows on the screen to perform various editing and scrolling functions. Moving into a window is called **inspecting** the window. The F1 key is also used to move forward between windows on the screen. The F2 key moves backward between windows. The escape key (<Esc>) exits inspect mode and returns to the command line.

When inspecting a window, different commands and functions are available that are not possible from the main command line. When a window is entered, the command and function key lines change to reflect operations specific to the selected window. On the left edge of the status line, a reminder is displayed indicating the contents of the window and your position within it.

See Section 7, Inspect Mode, for detailed descriptions of inspect mode for the various types of windows.

4.4 Escape Key

The escape key (<Esc>) can be pressed anywhere in the debugger to immediately terminate the current activity and return to the previous command level. If <Esc> is pressed while entering input in response to a prompt, one of two things can happen: If the prompt is a parameter to a command (such as the address at which to display memory), the entire command is immediately aborted and the response is ignored. If, however, the prompt is a setting or a value on the screen (such as a BTT qualifier or the value of a register in a window), the value is accepted and used before the debugger escapes to the previous level.

4.5 Edit Fields and Prompts

The debugger often requires you to enter some input in response to a prompt. The debugger has a simple field editor for this purpose. The field editor is also used in situations where you can move the cursor to a value on the screen and modify it, such as in inspect mode. The field editor has two different modes of operation, depending on the type of field being edited: **text** mode and **numeric** mode.

All command line prompts use the text mode. In text mode, all characters are accepted; the input can be any arbitrary expression or character string. The numeric mode is for direct modification of numeric values on the screen. If the value is displayed with suffix notation, the response must be a numeric value that also uses the suffix notation. The response can be in a different radix, so long as the radix is specified. Some values are displayed with an implied radix (see Section 3.5, Displayed Values). In this case, the only valid input for the field is a number in the same form, with no suffix allowed.

Examples:

1) Debugger prompts with:

```
"Display Memory - address: _______".
```

Valid response is any character string:

```
1000h    1234    10010001b    sym1    @(sym1 + 3)
```

2) Inspecting P registers, displayed value is "07Fh".
Valid input is any numeric constant:

```
0ffh     1234    1770h
```

3) Inspecting register file, displayed value is "7F".
Valid input is a hex constant only:

```
1f       20      3A
```

Using the field editor can be equated to building an output buffer from an input buffer. The input buffer contains the contents of the field before any editing and is either a default value supplied by the debugger or the current value of a displayed item. The input buffer is displayed on the screen when editing begins. As the output buffer is built, it overlays the input buffer on the screen. The output buffer is displayed with a different attribute (highlighted or reverse video) to differentiate it from the input buffer.

As characters are typed, they are added to the output buffer and the cursor advances to show the current position in the output buffer. Field editing can be terminated by pressing <CR> or <Esc>, or as shown in Table 4-1.

When editing terminates, the current contents of the output buffer are taken as the result. A special case occurs when the edit is terminated and there are no characters in the output buffer. In this case, the entire input buffer is copied into the output buffer and is used as the result. This way, the default or former value can be selected simply by pressing <CR> in response to the prompt.

Table 4-1 lists special control keys that can be used when editing fields. The effect of each key on the output buffer is stated for both text and numeric mode fields. Any control key not listed in the table, such as a function key, another arrow, etc., is ignored.

Table 4-1. Edit Field Control Keys

RETURN or **ESCAPE**	Terminates input and uses output buffer.
SPACE	Text: Inserts a space into output buffer. Numeric: Terminates input and uses output buffer.
BACKSPACE	Text: Erases character before cursor and backs up one space. Numeric: Backs up one space without erasing.
INSERT	Text: Subsequent characters are inserted at the cursor position and characters after the cursor are moved right, even out of the field. Remains in effect until the INSERT key or another control key is selected. Numeric: No effect.
DELETE	Text: Deletes character at cursor. Numeric: No effect.
LEFT ARROW	Backs up one space without erasing.
RIGHT ARROW	Copies the next character in the input buffer to the output buffer.
CTRL-LEFT ARROW	Text: Moves the cursor to the beginning of the output buffer. Numeric: Terminates input and uses output buffer.
CTRL-RIGHT ARROW	Text: Copies all characters to the right of the current cursor position from the input buffer to the output buffer. Numeric: Terminates input and uses output buffer.

4.6 Status Line

The second line of the screen is the status line. The debugger uses the left side of the status line to display special instructions or information. The left side of the status line is also used in inspect mode to indicate the contents of the current window and your position in the window.

The right side of the status line is used to indicate whether the CPU and BTT are currently running or halted. Whenever the CPU or BTT halt, the status codes are updated to show the reason for halting. In continuous run mode, the status of the CPU and BTT are displayed separately since they can be running or halted somewhat independently. Table 4-2 and Table 4-3 describe the status messages that appear on the right side of the status line.

Table 4-2. Status Codes

Reset Status Codes	
RESET	The emulator has been reset. This condition can be caused by hitting the reset button on the emulator or by the debugger resetting the emulator after it has responded improperly.
POWER-UP RESET	The emulator has just been powered up.
Status Codes in Normal Run Mode	
CPU/BTT RUNNING	The emulator is running.
STEPPING	The emulator is single-stepping.
HALTED: <halt reason>	The CPU has been halted for the reason shown. If no reason is given, the CPU has been halted by the user pressing a key.
Status Codes in Continuous Run Mode	
CPU: RUNNING **BTT: RUNNING**	The emulator is running.
CPU: RUNNING **BTT:** <halt reason>	The BTT has been halted for the reason shown. The CPU is still running. If the <halt reason> is simply *HALTED,* the BTT has been halted by the user.
CPU: <halt reason> **BTT: HALTED**	The CPU has been halted for the reason shown. If no reason is given, the CPU has been halted by the user.

Table 4-3. Halt Reasons

BPOINT	The CPU was halted by a simple breakpoint.
ACCESS VIOL	A memory protection violation occurred to halt the CPU.
HW BPOINT	The BTT was halted by a qualified hardware event.
TRACE FULL	The BTT was halted because the number of trace samples taken reached the specified maximum.
TIME OUT	The BTT free-running timer has reached a specified maximum time.
BAD OPCODE	The CPU halted because it tried to execute an illegal opcode.
SS DONE	The CPU halted after completing an instruction in single-step mode.

4.7 Updating the Screen

There are three ways the screen can be updated with new data from the emulator:

1) The debugger automatically updates the screen after the emulator has run and then halted.

2) Pressing F9 (Update) forces a screen update.

3) The debugger updates the screen at some other points, such as after the Register command.

A special case of a screen update occurs when the Execute command has been run from the trace sample screen. In this case, the debugger simply re-reads the trace buffer from the emulator and displays the most recent samples on the screen.

In the main screen, different windows are updated slightly differently. For screen updates after running, the code window is scrolled, if necessary, to position the current PC near the top of the window. For other screen updates, the code window is not scrolled.

If the PC remains in the virtual buffer for the code window, the disassembler is not invoked. This means that if you have self-modifying code, the modifications may not show up when the screen is updated. You can force a new disassembly in the code window by using F9 to update the screen, in which case the disassembler is always re-invoked.

Register and memory dump windows are updated by reading the displayed registers or memory from the emulator and changing the values on the screen. Any values that have changed since the last update are highlighted. The stack window is updated the same way, except that after running, the window is scrolled so that the top-of-stack is displayed at the top of the window.

The debugger updates the expression window by re-evaluating each expression in the window and displaying the new value. Values that have changed are highlighted.

Other windows, such as text, symbols, and module names are not updated since their data cannot be changed by the emulator.

Section 5

Top Level Commands and Functions

This section includes the following topics:

Table 5-1 summarizes the commands and functions available from the top level of the debugger.

Table 5-1. Top Level Commands and Functions

Command	Function
D (Display)	Fill display window with peripheral registers, memory, symbols, module names, or text file
X (Execute)	Run or single step
R (Register)	Display and modify a register
M (Memory)	Modify, Fill or Assemble memory
B (Breakpoint)	Set or remove breakpoints
E (Evaluate)	Evaluate any expression
C (Configure)	Configure debugger and memory map
T (Trace/Time)	Inspect trace samples
L (Load)	Load an object or configuration file
O (Module)	Set the current module
H (Halt)	Halt the CPU if running
S (System)	Temporary escape to operating system
Q (Quit)	Exit from debugger
F1 (Inspect)	Move to code window and enter inspect mode
F9 (Update)	Update screen with current data
F10 (Help)	Display help information

5.1 Fill Display Window - The D Command

The **D** (Display) command displays a menu that allows you to specify the contents of the display window as shown in Table 5-2.

Table 5-2. Display Commands and Functions

Command	Function
M (Memory)	Display a memory dump
P (Pregs)	Display peripheral registers
F (File)	Display a file
S (Symbols)	Display symbol table
O (Modules)	Display module names
C (Clear)	Clear window
ESC (Escape)	Return to top level
F1 (Inspect)	Inspect Window
F9 (Update)	Update Screen
F10 (Help)	Help for display command

The following sections describe each operation of the **D** command in detail. Many of the options prompt for parameters. After you specify what to display, the display window is cleared and re-filled with the specified data. If an error is detected and the window cannot be filled, it is left empty and the display submenu is redisplayed.

5.1.1 Display Memory - The M Option

You can display memory with the **M** option of the display command. The debugger prompts for the starting memory address:

```
Display Memory - address: _____
```

Enter an expression that represents the starting address. The debugger reads the contents of memory from the emulator and displays it in the window in a hexadecimal dump format. The address is displayed on the left of each line, followed by eight bytes of memory contents, followed by eight characters that represent the memory contents as ASCII values. Non-printable ASCII characters are represented by the period ('.') character.

You can move to the display window and scroll up and down through memory or modify individual addresses. See Section 7.3, Inspect Memory Dump.

5.1.2 Display Peripheral Registers - The P Option

You can display registers in the peripheral file with the **P** option of the display command. The debugger reads the contents of the registers starting at register P0 and displays them in the display window in a two-column hex format. Values are displayed with hex suffix notation.

You can move to the display window and scroll up and down through the peripheral registers, modify values, and change the display formats of individual registers. See Section 7.4, Inspect Peripheral Registers.

5.1.3 Display File - The F Option

You can show a text file in the display window using the **F** option of the display command. The debugger prompts you for a file name:

```
File: ____________
```

Type the pathname of the file to display. If the file exists, the debugger displays the first lines of it in the window. The maximum size of a file that can be displayed is 2048 lines. Any file larger than this is truncated to 2048 lines and displayed. Maximum file size is also limited by available memory, as the entire file is kept memory-resident.

You can move to the display window and scroll up and down through the file, temporarily expand the window so the file is displayed on the whole screen, and search the file for text strings. See also Section 7.5, Inspect File.

5.1.4 Display Symbols - The S Option

Use the **S** option of the display command to display symbols from the symbol table. The debugger prompts for two qualifiers:

```
Display Symbols - from module: _______ pattern: _______
```

The first prompt is for a module name. Use the word "all" to display symbols from all source modules or use a particular module name to display symbols from a particular module.

The second prompt is a template for symbol names to be displayed. Only symbol names that match the template are displayed. Two wildcard characters are allowed in the template: '*' which matches any string of characters and '?' which matches any single character. For example, the template "s*" would display only symbols that start with 's'. The template "s???s" would display only five letter symbols that start and end with 's'. The default template is simply "*", which causes all symbols to be displayed.

Symbols are listed in the display window with the symbol's module name on the left, the symbol name in the center, and the value (address) of the symbol on the right. A module name of "*" indicates the symbol is external.

You can move to the display window, scroll through the symbols, and search for text strings in the window. See Section 7.6, Inspect Symbols.

5.1.5 Display Modules - The O Option

The **O** option of the display command is used to list the names of all the source modules that are represented in the currently loaded executable object module. If the linker was not used to create the executable module, this command simply displays the name of the single source file. Otherwise, the names of all source modules that contributed object code to the executable module are listed in the display window. The debugger's current module, which is the one currently being used for all non-global references to the symbol table, is highlighted in the list.

You can move to the display window, scroll through the module names, search for text strings, and set any module to be the current module. See Section 7.7, Inspect Modules.

5.2 Execute Code - The X Command

The **X** (eXecute) command is used to cause the CPU to run. The command is unique in that it is available from three places in the debugger: at the top level, when inspecting code, or when inspecting trace samples. The eXecute command is is described in Section 6, Executing Code.

5.3 Display and Modify Registers - The R Command

Use the **R** (Register) command to display and modify the contents of one register. This command is a convenient way to change a register without having to move to the appropriate window. When you enter the **R** command, the debugger prompts for the name of the register:

```
Register: ____
```

Type the name of the register you want to display, such as *R12* or *R0C*. The debugger then reads the contents of the register from the emulator and displays it in hex format as the default response to a prompt:

```
value: <old value>
```

The contents of the register can be modified by typing an expression for the new value or left alone by simply pressing <CR> in response to the value prompt. The new value is written to the register, even if it wasn't changed.

The screen is updated so that any side-effects caused by modifying the register show up in the windows. See Section 4.7, Updating the Screen on page 4-8.

5.4 Modify, Fill or Assemble Memory - MM, MF and MA

The **M** (Memory) command allows either displaying and modifying the contents of one memory location or filling a range of memory with a particular value. A submenu is displayed; select **M** (Modify) or **F** (Fill).

If **M** is selected, the debugger prompts for a memory address:

```
location: _____
```

Type an expression representing the address of the memory location to display. The debugger then reads the contents of the memory location from the emulator and displays it in 1-byte hex format as the default response to a prompt:

```
value: <old value>
```

To modify the contents of the memory location, type an expression for the new value, or leave it alone by simply pressing <CR> in response to the value prompt. The new value is written to memory, even if not changed.

To fill memory, select the **F** option of the submenu. The debugger prompts for the address range and fill value as follows:

```
Fill Memory - start: _____ length: _____ value: _____
```

Enter expressions for the starting address and length to fill (in bytes). Then enter an expression representing a 1-byte fill value. This value is written to each address in the specified range. While memory is being filled, the debugger displays

```
Filling ...
```

on the status line. Fill operations can be somewhat lengthy; filling the entire address space (0000h - 0FFFFh) takes about 2.5 minutes.

To assemble a memory location, select the A option of the submenu. The debugger prompts for a memory address:

```
location: _____   asm:_____
```

Type an expression representing the address of the instruction to assemble. The debugger reads the location and displays the old disassembled instruction. Type in a new instruction followed by a <CR> to modify the location or just <CR> to leave it unchanged. If the instruction entered is invalid, an appropriate message will display and the location will be unchanged. Otherwise the instruction will be assembled and the proper opcode and operands written to memory beginning at the address entered.

> **Note:** No compensation is made for instances where the new instruction is not the same size as the original instruction (i.e., A two byte instruction to replace a one byte instruction will also overwrite the location following the given address).

After either a modify, assemble, or fill command, the screen is updated so that any resulting side-effects show up on the screen. See Section 4.7, Updating the Screen on page 4-8.

5.5 Breakpoint Operations - The B Command

The **B** (Breakpoint) command provides access to both types of breakpoints - simple and hardware. Hardware breakpoints (using the BTT) are handled by a special screen. This screen is accessed with the **B** (BTT) option of this command. Thus, access to the BTT screen is obtained by typing **BB** from the top level command menu.

Simple breakpoints are described in this section; hardware breakpoints with the BTT are described in Section 8, Hardware Breakpoints, Tracing and Timing. The options on the Breakpoint command submenu are described in Table 5-3.

Table 5-3. Breakpoint Commands and Functions

Command	Function
A (Add)	Add a simple breakpoint at a specified address
D (Delete)	Delete a simple breakpoint
R (Remove All)	Remove all simple breakpoints
L (Load)	Load breakpoints from file
S (Save)	Save breakpoints to a file
B (BTT)	Go to the BTT screen for hardware breakpoints
ESC (Escape)	Return to the top level command menu
F10 (Help)	Help for the Breakpoint command

5.5.1 Simple Breakpoints

Simple breakpoints are set on memory addresses with no other qualifiers. If the CPU is running and an instruction acquisition occurs at an address where a simple breakpoint has been set, the CPU halts prior to fetching the next instruction. Simple breakpoints are set in code, so that an instruction fetch at some address stops the CPU. Note that the CPU halts AFTER the instruction at which the breakpoint is set. Also, note that a simple breakpoint encountered in continuous run mode does halt the CPU, unlike hardware breakpoints which halt only the BTT.

You can have up to 30 simple breakpoints set at any time. Each breakpoint has a breakpoint ID number for reference. ID numbers start at 1 and are allocated as they are available. For example, the first breakpoint set is assigned an ID of 1. The next is assigned 2, then 3, etc. If breakpoint 2 is cleared, ID 2 becomes available and the next breakpoint is assigned an ID of 2.

In the code window, addresses where simple breakpoints have been set are identified with the ID number of the breakpoint displayed just to the left of the address. You can also easily set and remove simple breakpoints while inspecting the code window. See Toggle a Simple Breakpoint - The B Command on page 7-6.

5.5.2 Setting a Simple Breakpoint - The A Option

Use the **A** (Add) option of the Breakpoint command to set a simple breakpoint. The debugger prompts for the address where the breakpoint is to be set:

```
Address: ________
```

Type an expression representing the address where you want the breakpoint. If the breakpoint is on an address of an instruction in the code window, the reference number of the breakpoint appears in the window next to the instruction.

5.5.3 Removing a Simple Breakpoint - The D Option

The **D** (Delete) option of the Breakpoint command removes a breakpoint with a specified ID number. The debugger prompts for the ID of the breakpoint to be removed:

```
Remove breakpoint #: ____
```

Enter the number of the breakpoint to be removed. If the ID number represents a breakpoint that is currently set, the breakpoint is removed. Otherwise, the debugger displays an error message to indicate that no breakpoint with that ID has been set.

5.5.4 Removing all Simple Breakpoints - The R Option

The **R** (Remove all) option simply removes all simple breakpoints that are currently set.

5.5.5 Loading a Breakpoint File - The L Option

The **L** (Load file) option loads a previously saved breakpoint file. The debugger prompts you for a filename; you may omit the extension if it is ".bp", the default for breakpoint files.

Breakpoint files can also be loaded using the **L** (Load) command at the top level debugger menu; see Section 5.9 on page 5-20.

5.5.6 Saving a Breakpoint File - The S Option

The **S** (Save file) option saves the current breakpoint settings in a disk file for later retrieval. This allows the use of different breakpoint settings for different debugging sessions without having to manually re-enter the breakpoints each time.

The debugger prompts for a filename; the default extension is ".bp" and should be used so that the file can be loaded from the main screen also. If a file with the same name already exists, it will be replaced by this file.

5.6 Evaluate an Expression - The E Command

The **E** (Evaluate) command provides a way to evaluate an expression and display the result. You can use it to see the contents of a variable, do arithmetic, or convert between hex and decimal. You can also add the expression to the expression window for permanent tracking.

The debugger prompts you for the expression to evaluate:

```
Expression: _____
```

Any expression (see page 3-6, Expressions) entered is evaluated by the debugger and displayed as a 16-bit value in both hex and decimal. For example, suppose you type the expression "PC + 23h" and the PC currently contains the value 7000h. The debugger displays

```
PC+23h = 07023h = 28707
```

and displays the message

```
<CR> to save, <ESC> to exit
```

Pressing <CR> adds the expression to the bottom of the expression window in hex format and word data size (see Inspect Expressions, page 7-23). If there is no need to remember the value of the expression, press <Esc>.

For the purposes of expression evaluation, remember that:

- Register A is an alias for R00.
- Register B is an alias for R01.
- Two-byte values are located in memory at addresses [x-1]:[x], where [x] contains the low-order byte.
- symbol = value of symbol.
  #symbol = value of symbol (used in order to be consistent with assembler).
  *symbol = contents of symbol. value = ([symbol-1]:[symbol])
  @symbol = value at address pointed to by contents of symbol; same as **symbol. value = (([symbol-1]:[symbol]))

Refer to Table 3-1 on page 3-7 for a list of valid operators in expressions.

Table 5-4 shows several different kinds of expressions and how they are evaluated by the debugger. The examples shown use the following code and memory assignments, with any unspecified memory set to zero.

```
                        FLAGS   .EQU    R016
                                .sect   mem0,2040h
    2040    8C11        TABLE   .WORD   8C11h

                                .sect   mem1,7000h
    7000    52F0        START:  MOV     #0F0h,B

Register 15 (R0F)  70h   Register 21 (R015)  20h   Memory 8C10h   08h
Register 16 (R010) 01h   Register 22 (R016)  41h   Memory 8C11h   45h
```

The debugger evaluates the following expressions as shown, assuming a default display radix of hexadecimal and word display length.

Table 5-4. Expression Evaluation Examples

Expr	Comment	Value	Notes
16	Immediate value	0010h	Value of the number
#16	Immediate value	0010h	Value of the number
*16	Contents of value	7001h	Contents at address
@16	Word indirect	52F0h	Value pointed to by contents at address
R016	Register value	0016h	Value of register, hexadecimal
#R016	Register value	0016h	Value of register
*R016	Register contents	2041h	Contents of register (R015:R016)
@R016	Register indirect	8C11h	Value pointed to by register ((R015:R016))
*@R016	Address indirect	0845h	Value pointed to by address in location pointed to by contents of register (((R015:R016)))
FLAGS	Defined register	0016h	Value of defined register
#FLAGS	Immediate value	0016h	Value of defined register
*FLAGS	Immediate value	2041h	Contents of register
@FLAGS	Register indirect	8C11h	Value pointed to by contents of register
TABLE	Defined symbol	2040h	Value of program variable 'TABLE'
#TABLE+1	Symbol address	2041h	Value of program variable 'TABLE'+1
*(TABLE+1)	Symbol address	8C11h	Contents at address TABLE:TABLE+1
@(TABLE+1)	Symbol pointer	0845h	Contents at address ((TABLE:TABLE+1))
START	Code Label	7000h	Value of symbol or label
#START	Symbol address	7000h	Value of symbol or label
*(START+1)	Word indirect	52F0h	Contents of address; (7000h)

5.7 Configuration and Memory Map - The C Command

The **C** (Config) command provides access to the device configuration and memory mapping functions of the debugger. These functions use a special screen that is drawn when the 'C' command is chosen. This screen is fully described in Section 9, Debugger Configuration. Memory mapping is described in Section 10.

5.8 Inspect Trace Samples - The T Command

The **T** (Trace/Time) command is used to display trace samples that have been collected by the BTT board and also to display the values of the two BTT timers. When you type **T**, the main screen is erased and replaced with the trace sample screen. Most of the screen consists of space for the trace samples. A window at the bottom of the screen displays the timer values. This window can be toggled on and off the screen with the **I** (tImers) command, so that you have more lines available for trace samples if you don't need to see the timers.

The Trace command is actually an *inspect* function – it puts you in a window where you can move and scroll through trace samples in much the same way that you inspect the other windows.

See Figure 5-1 for an example of the trace sample screen.

```
Inspect Trace: Position Top Bottom Lookup Save Timers FormatTime eXecute
Sample = 0, Total = 12
INDX ST   h  m  s  ms  us  ns  EXTERNAL CYCLE ADDR DATA REVERSE ASM
0000 0    0:00:00.000 000 400 11111111 IAQ   7000  52  MOV   #02,B
0001 0 E  0:00:00.000 000 800 11111111 READ  7001  02
0002 0    0:00:00.000 001 000 11111111 READ  0001  02
0003 0    0:00:00.000 001 200 11111111 WRITE 0001  02
0004 0    0:00:00.000 001 600 11111111 IAQ   7002  FD  LDSP
0005 0    0:00:00.000 002 200 11111111 READ  0004  06
0006 0    0:00:00.000 002 400 11111111 READ  0001  02
0007 0    0:00:00.000 003 000 11111111 IAQ   7003  8E  CALL  clear
0008 0    0:00:00.000 003 400 11111111 READ  7004  70
0009 0    0:00:00.000 003 800 11111111 READ  7005  0B
0010 0    0:00:00.000 004 400 11111111 WRITE 0003  70
0011 0    0:00:00.000 004 800 11111111 WRITE 0004  06

        timer 1: 0:00:00.000 000 000      timer 2: 0:00:00.000 000 000
     timer 1 avg: 0:00:00.000 000 000
          F3PgDown F4PgUp                    F1Help
```

Figure 5-1. Trace Sample Screen

5.8.1 Trace Samples

Trace samples are snapshots of bus cycle activity that are collected and stored by the BTT while the emulator is running. The BTT must be programmed to collect trace samples. Programming the BTT is described in Section 8, Hardware Breakpoints, Tracing and Timing. The BTT has a circular trace buffer that can hold 2047 samples. If more samples than this are collected, the buffer wraps around and new samples overwrite the old ones. Each sample is 104 bits wide and contains the following information:

- Address and data bus values
- Bus cycle type
- External logic probe values
- State and event indicators
- A time stamp from the free-running timer.

The state and event flags indicate which state the BTT was in when the sample was taken, and whether an event or breakpoint occurred on the cycle represented by the trace sample.

Each trace sample is referred to by its index in the trace buffer, starting from 0. Trace samples with lower indices are chronologically older than samples with higher indices. Thus, if the buffer is full, sample 2046 is always the most recent sample taken.

When trace samples are displayed on the screen, they include the information shown in Table 5-5, identified by a heading displayed on top of the screen.

Table 5-5. Displayed Trace Samples

Field	Description
INDEX	Index in trace buffer
STATE	BTT state in which sample was taken
FLAG	E - event occurred on this cycle L - last event (cycle caused breakpoint)
TIME STAMP	Timer value, displayed in the format: hh:mm:ss.mmm uuu nnn where hh = hours mm = minutes ss = seconds mmm = milliseconds uuu = microseconds nnn = nanoseconds
EXTERNAL	Values of external probe inputs
CYCLE TYPE	READ - memory read cycle WRITE - memory write cycle IAQ - instruction fetch cycle
ADDRESS	16-bit value of address bus
DATA	8-bit value of data bus
REVERSE ASM	Disassembled code, for IAQ cycles

On IAQ cycles, the trace buffer includes a reverse assembly of the instruction that was executed.

IAQ samples in the trace buffer are displayed in a different color in order to distinguish the executed instructions from the rest of the memory cycles. Time stamps are also displayed in a different color.

5.8.2 Timers Window

The window at the bottom of the trace screen displays the value of the two programmable BTT timers. These timers can be programmed to time the intervals between specific events in your program or collect performance statistics. Programming the timers is described in Section 8, Hardware Breakpoints, Tracing and Timing.

The BTT has two separate programmable timers. The current setting for timer #1 is displayed on the left side of the window, and the setting for timer #2 is displayed on the right.

A third time value, representing the average time for timer #1, is displayed on the left underneath the setting for timer #1. Average time is defined as the total accumulated time, divided by the number of times the timer was started. For example, if timer #1 has an accumulated time value of 10.0 seconds, and it was started 20 different times, the average time would be 0.5 seconds. Average time is computed for timer #1 only; the BTT does not have the capability to determine the average time for timer #2.

5.8.3 Inspect Trace Commands

When the trace screen is first entered, the debugger checks the emulator to determine how many trace samples have been collected. If the emulator has no trace samples, the message

```
NO TRACE SAMPLES TAKEN
```

appears on the screen. Otherwise, the debugger reads a screenful of the most recent trace samples and displays them on the screen. A menu of special commands for trace samples is displayed, along with a special set of function keys. The commands and functions available while inspecting trace samples are shown in Table 5-6.

Table 5-6. Inspect Trace Commands

Command	Function
P (Position)	Position screen at specified index
T (Top)	Position screen at top (oldest samples)
B (Bottom)	Position screen at bottom (newest samples)
L (Lookup)	Find a sample with certain qualifications
S (Save)	Save samples to a file
I (Timers)	Toggle the timers window on or off the screen
F (Format)	Select a format for the time stamp
X (Execute)	Run the CPU to collect more samples
F1 (Find Next)	Lookup next sample matching previous qualifications
F3 (Down)	Scroll down through samples
F4 (Up)	Scroll up through samples
F10 (Help)	Help for inspect trace samples
ESC (Escape)	Return to main screen and top level command menu

The arrow keys can be used in addition to the keys listed above to move up and down through the trace buffer and left and right on the screen.

5.8.4 Position at an Index - The P Command

The **P** (Sample Number) command in the inspect trace screen is used to position the screen at a particular sample index. The debugger prompts for the desired sample number:

```
Sample Number: _____
```

Enter a number from 0 to the maximum index, which is the total number of samples (shown on the status line) minus 1. The window scrolls or refills, if necessary, and the cursor moves to the sample with the specified index.

5.8.5 Position at the Top Samples - The T Command

Use the **T**(op Samples) command to position the screen at the top of the trace buffer. The oldest samples in the buffer are displayed, starting with index 0.

5.8.6 Position at the Bottom Samples - The B Command

Use the **B**(ottom Samples) command to position the screen at the bottom of the trace buffer. The most recent samples in the buffer are displayed. If the buffer is full, the highest index is 2046. Otherwise, the highest index is one less than the total number of samples taken, which is indicated on the status line.

5.8.7 Look for a Qualified Sample - The L Command

The **L** (Lookup) command allows you to search the trace buffer for a sample that matches a particular set of qualifications. The qualifications are specified in much the same way as for an action in the BTT.

When you type **L**, a small window opens on the right side of the screen that contains a list of qualifiers. These qualifiers are identical to the qualifiers used to specify a BTT action. The function keys and control keys are also used in exactly the same way. See Section 8.7, Editing Actions, for a description of how to enter these qualifiers. One additional qualifier is included after the others: a *Flag* parameter that allows you to specify *EVENT, LAST,* or *NONE* (use the space bar to toggle though these selections).

When you are finished specifying qualifiers, press either F1 (Accept) or <Esc> to begin the search. The emulator begins at the current cursor position and searches forward through the buffer for a sample that matches the specified qualifiers. If one is found, the screen is scrolled so that the matching sample is on the screen. The matching sample will be highlighted. If no matching sample is found, the message "not found" is displayed, and the cursor remains at its present position.

To search for the next sample with the same qualifications, press F1 again. This leaves the qualifiers unchanged for use in the next search.

5.8.8 Save Trace Samples in File - The S Command

The **S** (Save) command on the trace screen saves trace samples to a disk file for later inspection. ANY number of samples can be saved, up to the number that were recorded. When you enter **S**, the debugger prompts for the name of the file to which the samples will be written:

```
Save trace samples in file: _____
```

After the filename is entered, the debugger prompts for the starting and ending sample numbers:

```
start: ____        end: ____
```

Enter the indices of the first and last samples to be saved. The default start index is 0, and the default end index is the last sample taken; using both defaults causes the whole trace buffer to be saved.

The specified trace samples are read from the emulator, formatted just as they would be for display on the screen, and written to the file. If many samples are saved, this process can be somewhat lengthy. For example, saving a full trace buffer of 2047 samples takes approximately 2.5 minutes.

5.8.9 Toggle the Timers Window - The I Command

The **I** (tImers) command removes and restores the timers window from the bottom of the screen, which expands the screen space by four trace samples. This is a convenient command if the timers are not being used, because it allows more trace samples to be seen at one time.

If the timers are used later in the debugging session, simply type **I** again to restore the timers window.

5.8.10 Format the Time Stamp - The F Command

The Time Stamp in the trace samples can be displayed in one of three different ways, allowing you to more easily comprehend the data according to your needs. Use the **F**(ormat) command to select one of the three possible formats:

Absolute Display time elapsed since beginning the execution of this run.

Delta Display the difference between each sample and the previous sample.

Mark Display the time from the current sample. Samples before the current sample will show a negative time and samples after the current sample will show a positive time.

5.8.11 Execute Code from the Trace Screen - The X Command

Code can be executed while inspecting trace samples using the **X** (eXecute) command from the inspect trace menu. The Execute command operates here in exactly the same way as it does from the top level command menu. The command is described in Section 6, Executing Code.

After code executes and the BTT halts, the trace screen is updated by reading new trace samples from the emulator and displaying the most recent ones. This command is especially useful in continuous run mode, where you can repeatedly trace program operation as it executes without ever leaving the trace screen.

5.9 Loading a File - The L Command

The **L** (Load) command loads a disk file into the debugger. There are several different types of files that may be loaded; this same command is used in exactly the same way to load any of them. The debugger prompts for the name of the file to be loaded:

```
Load file: _____
```

Type the pathname of the file to be loaded. The debugger uses the extension of the file name to determine the type of file being loaded. In order to load files using this command, the naming conventions for extensions must be followed. Table 5-7 lists the various file types and the extensions used to identify them.

Table 5-7. Conventions for File Extensions

Extension	File Type
.out	Object file containing an executable TMS370 program
.mmp	Memory map file
.bp	Simple (software) breakpoint file
.btt	BTT configuration and hardware breakpoint file
.exp	Expression file
.cmd	Command file (used to load other files)

The debugger identifies the file type and looks for the file. If the file is found, the debugger loads it as described below. A message identifying the file name and type, such as

```
Loading object file "test.out"
```

is displayed on the status line to indicate that the file is being loaded. Files with an extension not listed above are loaded as object files (see the following section). If the file cannot be found, an error message is displayed.

5.9.1 Loading an Object File

If the filename has an extension of ".out" or has no extension at all, the debugger loads it as an object file. ".Out" is appended to any filename that has no extension, unless the filename ends with a period ('.'). For example, the debugger assumes the following filenames represent object files:

test.out	
test	(loaded file is *test.out*)
test.	(loaded file is *test*)

The TMS370 linker produces an executable object file. Loading the executable file is the first step to take when using the debugger to run a program. The debugger loads symbolic information from the file into its internal symbol table and object data from the file into target memory at the addresses specified in the file. No relocation is performed when the file is loaded. Any symbols that were in the symbol table from a previous file are erased; the symbol table contains only symbols from the currently loaded file.

Whenever an object file is loaded into the debugger, the expression window is cleared since some expressions may depend on symbols that are no longer in the symbol table. Also, all simple breakpoints are cleared. The PC is set to the entry point defined in the file, and the screen is completely updated.

5.9.2 Loading Configuration Files

The Load command at the top level can also be used to load configuration files for the debugger, provided that the naming conventions for filenames are followed. You can load memory map files, breakpoint files, BTT setup files, and expression files. These configuration files are created using individual *Save* commands within the various local command menus where you set up the configurations themselves. For example, a memory map save file is created from the memory map screen using the **S** command. Configuration files can be also be loaded from the local menus using individual *Load* commands. The **L** command at the top level simply provides an alternative way for you to load the files, without having to go into one or two levels of subcommands. Configuration files are loaded in exactly the same way, regardless of where you invoke the command.

Memory Map configuration files are described in Section 10.3, Top Level Memory Map Commands. BTT files are described in Load a BTT Configuration - The L Command on page 8-16 and Save a BTT Configuration - The S Command on page 8-17. Section 7.11.5 and Section 7.11.6 starting on page 7-25 describe saving and loading of expression files.

5.9.3 Command Files

The debugger has a very simple **batch** capability that allows it to load object and configuration files under the control of a special file called a **command file**. The command file contains commands to load other files. Thus, by using a single command, you can cause several files to be loaded. You can set up a custom command file for a particular debugging application; the command file would load your object, BTT, memory map, simple breakpoints, and expression files. Then, simply load the command file when you invoke the debugger, thereby avoiding tedious initialization and set-up chores. The file extension of the command file must be ".cmd".

Each line of the command file must contain the word **load**, followed by the name of a file to load. The filename may contain a disk drive designator and a pathname, if the file is to be loaded from a different directory. Blank lines are allowed in command files. Command files must not contain anything except commands; any extraneous text or poorly formed commands will cause the following message

```
command not recognized -- hit any key
```

to be displayed when the file is loaded.

A file type keyword can be used between the word **load** and the filename to override the default file type, which would normally be determined from the extension.

Table 5-8. Command File Example

Assume the following command file:

```
load c:\test\test.mmp
load test
load btt \test\bpoint.dat
load test.exp
```

Loading this file causes the following files to be loaded:

A memory map file `test.mmp` from the directory `c:\test`

An object file `test.out` from the current directory

A BTT file `bpoint.dat` from the directory `\test`

An expressions file `test.exp` from the current directory

Note the use of the keyword `btt` to identify the file `bpoint.dat` as a BTT file. The file type keywords are the same as the default extensions.

Each file is loaded just as if you used the **L** command to load it individually. The status line displays a normal "`loading ...`" message as each file is loaded, although some files are loaded so quickly that the message is displayed for only a fraction of a second.

Remember that loading an object file clears all expressions and simple breakpoints so, in general, object files should be specified first in the command file before any expression or simple breakpoint files.

5.9.4 Loading Files from the Command Line

When invoking the debugger from the command line of DOS, you can include any number of filename arguments. Files specified on the command line are automatically loaded when the debugger comes up. For example, suppose the debugger is invoked with the command

```
C>  dbr370 test.mmp test test.btt
```

The object file `test.out`, the BTT file `test.btt`, and the memory map file `test.mmp` are immediately loaded as soon as the debugger initializes. Command files can also be loaded in this way.

5.10 Setting the Current Module - The O Command

The **O** (mOdule) command is used to set the current module name in the debugger. The current module determines how non-external symbols are to be accessed. When you use a symbol in an expression, if the symbol is not an externally defined symbol, it must be defined in the current module or the debugger does not recognize the symbol. You can use the **O** command to specify the name of the source file that will be used as the current module. The debugger prompts for the module name, using the current module name as a default.

```
Current Module: ________
```

Enter the filename. If the filename is in the list of source files that were used to build the executable file, the current module is set to the specified file. If no symbol table has been loaded, or the module you typed is not part of the object file, an error message is displayed and the current module remains unchanged.

You can display the list of modules using the **D** (Display) command and the **O** option from the top level command menu. See Display Modules - The O Option on page 5-4.

5.11 Halting the CPU - The H Command

The **H** command halts the CPU. If the CPU is not running, the message

```
Not running
```

is displayed and the command is ignored. Otherwise, the CPU is halted and the screen is updated. See Section 4.7, Updating the Screen.

The Halt command is the only way you can force the CPU to halt after running it in continuous run mode.

5.12 Escaping to DOS - The S Command

If you type the **S** (System) command from the top level menu, the debugger suspends itself, remains resident in memory, and exits to the operating system. The screen is cleared, the message

```
Type EXIT to return to debugger
```

is displayed, and DOS is loaded. You can perform as many DOS operations as the memory allows, since the debugger remains resident in memory. When you are ready to return to the debugger, type `exit` at the DOS prompt. The debugger screen will be restored and you can continue using the debugger normally.

The System command has no effect on the debugger or emulator environment. In fact, the CPU can be running during the whole time you escape to DOS. It will not be affected when you exit or return to the debugger.

If the PC does not have enough memory to save the debugger, the system will return to the debugger. If this happens, save all needed data and execute the **Q**uit command.

5.13 Leaving the Debugger - The Q Command

The **Q** (Quit) command is for exiting the debugger. Before the debugger actually quits, you must confirm that this is what you want to do. The debugger prompts you with:

```
Confirm: ___
```

Type **Y** or **y** to confirm that you really want to quit. Any other key is interpreted as a retraction of the quit command - the command is ignored and the debugger returns to the top level command menu. Note that the default for the confirm prompt is **y**, so if you simply press <CR>, you will quit.

When the debugger quits, it does not affect the emulator at all. If the emulator is running, it will continue to do so. However, when the debugger is invoked next time, the emulator will be reset so that the debugger can initialize it.

5.14 Moving to Windows - F1

You can move from the top level command menu into the windows on the screen to perform various editing and scrolling functions. This is called *inspecting* the windows. Use the F1 (Inspect) key to go into inspect mode. The cursor moves into the code window and the menu and function key lines are replaced with commands specific to inspecting code. Once in inspect mode, the F1 key moves forward through the windows on the screen, and the F2 key moves backwards through the windows. Press <Esc> to return from inspecting a window back to the top level command menu.

Refer to Section 7, Inspect Mode, for information on the operations you can perform in the various windows while in inspect mode.

5.15 Updating the Screen - F9

The F9 (Update) key forces the debugger to update all displayed values from the emulator and redisplay them on the screen. This is useful after you have performed some operation that may have caused displayed values to change and you wish to see exactly what changed as a result of the operation. The process by which the debugger updates the screen is described in Section 4.7, Updating the Screen.

5.16 On-Line Help - F10

The F10 (Help) key at the top level invokes the top level of the debugger's on-line help utility. When help is invoked, a temporary window opens on the screen and help text for the main menu is displayed. You can invoke additional layers of help by following the instructions given in the help text. When you exit help, the temporary window is removed and operation proceeds normally.

> **Note:**
>
> The help file `DBR370.HLP` must be in either the current directory or the directory pointed to by the DOS environment variable `IPCDIR`.

Section 6

Executing Code

The **X** (eXecute) command allows you to run your programs with the TMS370. It can be accessed from the top level command menu, the code window, or the trace sample screen. The Execute command has two forms. The form used depends on whether the current run mode is **normal** or **continuous**.

Both forms of the execute command operate in the same general way. First, a submenu is displayed that allows you to select how the CPU is to be run (see Table 6-1 on the next page). After all the selections have been made, the CPU and BTT start running. The debugger displays a message on the status line of the screen to indicate that the emulator is running and waits for one of the following events to occur:

- A key is pressed on the keyboard.
- The emulator (CPU or BTT or both) stops on its own.

The emulator may stop because of a breakpoint, completion of a single step operation, access violation, or some other exception condition. Whenever the emulator halts, the emulator status display on the right side of the status line is updated to indicate the reason. If the emulator is running, the display also indicates this. See Section 4.6, Status Line, on page 4-6 for a description of the status line codes.

The action taken by the debugger when either a key is pressed or the emulator halts depends on the run mode and the actual execute option specified.

This section includes the following topics:

6.1 Execute Command Options

The submenu of the execute command selects how the CPU is to be run. The single-step options in Table 6-1 are available only in the submenu for the normal run mode. Single-stepping is not possible in the continuous run mode. Detailed descriptions of the execute options follow the table.

Table 6-1. Execute Command Options

Options For Normal Run Mode Only	
Command	**Function**
I (Instruction)	Single-step through one or more instructions
S (Statement)	Single-step through instructions and function calls
L (Loop)	Single-step once through a loop
W (While)	Single-step while an expression is true
U (Until)	Single-step until an expression is true
F (Function)	Single-step to the end of a function

Options For All Run Modes	
Command	**Function**
G (Go)	Start running from current PC
R (Reset)	Software reset and run
A (tArget Reset)	Wait for target to be reset and run
ESC (Escape)	Return back to top level command menu
F10 (Help)	Help for execute command
T (Trace)	Inspect trace samples

6.1.1 Single-Stepping - The I Option

Use the **I** (Instruction) option for single-stepping. The debugger prompts for the number of instructions to execute:

```
Execute Instruction--how many: _____
```

The emulator begins single stepping through the specified number of instructions.

6.1.2 Single-Stepping - The S Option

The **S** (Statement) option is similar to the instruction option. The difference is in the way CALL instructions are treated. When executing a CALL instruction, the entire function call is treated as one statement. Using the statement option, you can step over an entire function call without stepping through the entire function. For example, in the following situation, executing one statement would execute *func* and leave the PC at 7003.

```
PC -->  7000  CALL  func
        7003  MOV   A,B
```

The statement option also prompts for a count of how many statements to execute.

6.1.3 Executing a Loop - The L Option

The **L** (Loop) option repeatedly single-steps until the PC returns to the current position. Using the loop option allows easy execution of one iteration of a loop.

> **Warning:**
>
> **Do not use this option if your program is not inside a loop.**

6.1.4 Conditional Execution - The W Option

The **W** (While) option of the execute command prompts for a conditional expression:

```
Execute While-condition: ————————
```

The expression is evaluated. If the result is true (nonzero), the CPU single-steps through one instruction and the expression is re-evaluated. This evaluation/single-step cycle is repeated as long as the value of the expression is nonzero. For example, in the following situation, executing while `B>5` would run through the loop 5 times and stop with the PC at 7002 and a value of 5 in register B.

```
PC -->  7000  MOV   #10,B
        7002  DJNZ  B,7002
```

Refer to Table 3-1 and Table 5-4 for a list of expression operators and notes on expression evaluation.

6.1.5 Conditional Execution - The U Option

The **U** (Until) option is similar to the While option. However, the conditional expression is evaluated after each single-step rather than before. Also, execution continues as long as the expression evaluates to false and halts when the expression becomes true.

Refer to Table 3-1 and Table 5-4 for a list of expression operators and notes on expression evaluation.

6.1.6 Returning to Caller - The F Option

The **F** (Function) option repeatedly single-steps until the RTS or RTI instruction that ends the subroutine is executed; thus, it will step through a subroutine within a subroutine. When debugging a function, this option can be used to finish executing the current function and return to the calling function.

6.1.7 Run from Current PC - The G Option

The **G** (Go) option is the standard "run" command. It simply starts the CPU running from the current PC.

6.1.8 Software Reset and Run - The R Option

The **R** (Reset) option asserts the reset line on the TMS370 device. The device fetches the reset vector from memory, loads the vector into the PC, and begins running. This option always causes the BTT to be reset.

6.1.9 Wait for Target Reset - The A Option

The **A** (tArget Reset) option is like the Reset option but instead waits for the reset signal to be asserted externally from the target application system through the target cable. The device then fetches the reset vector and begins running. This option allows you to exercise the reset circuitry of your system.

6.1.10 Inspect Trace Samples - The T Option

See Section 5.8 for details.

6.2 Running in Normal Mode

In normal run mode, depending on the execute option selected, the emulator either begins single-stepping or running. The right side of the status line shows either STEPPING or CPU/BTT RUNNING to indicate what the emulator is doing.

If the emulator is **STEPPING**, the left side of the status line displays the following message.

```
Hit any key to halt
```

The emulator continues single stepping until either a key is pressed or the specified single-step operation terminates. The screen is then updated (see Section 4.7, Updating the Screen).

If the emulator is **RUNNING**, the left side of the status line displays the following message.

```
Press 'C' to continue or any other key to halt
```

The emulator continues running until a key other than 'C' is pressed or it halts on its own. When the emulator halts, the CPU and BTT halt together. The debugger updates the screen (see Section 4.7, Updating the Screen), re-displays the submenu of execute options, and displays the following message.

```
<SP> = repeat
```

You can press the space bar to repeat the last execute option (useful for single-stepping), select another execute option, or press <Esc> to return to the top level command menu.

Pressing the **C** key while the emulator is running causes the debugger to immediately return to the top level command menu. Other operations can be performed while the emulator continues to run. However, no operation can be performed that affects the emulator in any way. Trying such an operation causes the following error message to be displayed.

```
Can't perform the command, emulator is running
```

You can, however, display a file, symbols, or modules, escape to the operating system, or anything else that does not require access to the device.

The emulator may halt on its own or be explicitly halted using the **H** (Halt) command from the top level menu. In either case, the status line indicates the reason for halting (see Section 4.6, Status Line, on page 4-6) and the screen is updated.

6.3 Running in Continuous Mode

In continuous mode, the BTT can be stopped independently of the CPU and then re-started. The emulator cannot single-step in continuous mode.

If the CPU is already running when the **X** command is entered, the command simply starts running the BTT. If the CPU is not running, an option must be selected from the execute submenu. This submenu is like the submenu for normal run mode with the single-step commands omitted.

The CPU is started according to the selected option (unless it was already running), and the BTT is also started. The right side of the status line indicates this condition by showing: `CPU: RUNNING BTT: RUNNING`. CPU status and BTT status are always displayed separately in continuous run mode.

The debugger displays the following message:

```
Hit any key to halt BTT
```

and waits for the BTT to be halted in one of three ways: any key is pressed, the BTT halts on its own, or the CPU halts on its own and forces the BTT to halt.

The right side of the status line indicates what caused the BTT or CPU to halt. (See Status Line, page 4-6.) After the BTT halts, the screen is updated and the debugger returns to the either the top level command menu or the inspect trace menu, depending on where you were when the **X** command was entered.

Section 7

Inspect Mode

Inspect mode allows you to perform various editing, scrolling, and other operations on the information displayed in the windows of the screen. Each window has its own unique combination of operations that can be performed during inspection. However, operations are consistent between windows; similar functions in different windows are generally invoked the same way.

This section describes the features common to inspecting all the windows and the meaning of each of the inspect mode function keys. Following sections provide specific information on inspecting each type of window in the debugger.

Many of the commands and functions in inspect mode operate on a value or object at the current cursor position. When a window is entered for inspection, the debugger moves the cursor into the window. The cursor can then be moved around in the window with the commands and functions described in this section.

The debugger uses the status line as a current position indicator to help keep track of where you are on the screen and what is at the current cursor position. The left side of the status line displays a field that describes what window you are in and the address or line number of where you are within the window. Whenever you move in the window, the status line is updated to reflect your new position.

Inspect mode relies heavily on function keys for various operations. The same set of function keys is used for inspecting all windows, although in some windows certain keys are not appropriate and are therefore disabled. Table 7-1 on the next page describes the inspect mode functions and the operations they perform. Specific information about the function keys for each window type is given in the sections that describe the windows themselves.

This section includes the following topics:

7.1 Function Key Usage

Table 7-1 describes the usage of function keys while in Inspect mode.

Table 7-1. Inspect Mode Function Keys

Command	Function
F1 (Next)	Move to next window and inspect it
F2 (Previous)	Move to previous window and inspect it
F3 (Down)	Scroll forward
F4 (Up)	Scroll back
F5 (Dump)	Memory dump using current cursor value
F6 (DisAsm)	Disassemble using current cursor value
F8 (Expand)	Expand window to full screen
F9 (Update)	Update the screen
F10 (Help)	Help for current window
ESC (Escape)	Leave inspect mode and return to top level command menu

Note:

On the TI PC host, the Shift-DOWN ARROW and Shift-UP ARROW keys have the same effect as the F3 and F4 function keys. On the IBM PC host, the PAGE-DOWN and PAGE-UP keys on the keypad have the same effect as F3 and F4.

In addition to the function keys listed in Table 7-1, certain control keys are used in various windows to move the cursor. These keys are listed in Table 7-2.

Table 7-2. Inspect Mode Control Keys

Command	Function
L ARROW	Move cursor left one space or item
R ARROW	Move cursor right one space or item
UP ARROW	Move up one line, scrolling if necessary
DN ARROW	Move down one line, scrolling if necessary.
SPACE	Move cursor right one space or item
INSERT	Scroll text windows left
DELETE	Scroll text windows right
RETURN	Exit edit mode

7.1.1 Moving Between Windows - F1 and F2

The F1 (Next) and F2 (Previous) keys are used in all the inspect windows to leave the current window. F1 moves to the next window on screen, and F2 moves back to the previous window. The order of windows on the screen for moving between them is:

1) Code
2) Display (Memory, P registers, File, Symbols, or Modules)
3) CPU Registers
4) Register File
5) Stack
6) Expressions

7.1.2 Scrolling - F3 and F4

The F3 (Down) and F4 (Up) keys are used in all but the CPU Registers window to move the cursor down or up by the length of the window. The window is scrolled if possible and re-filled with either data from the window's virtual buffer or new data from the emulator. When the top or bottom limit of a window is reached, such as the beginning or end of a file, the window does not scroll further and these function keys are ignored.

7.1.3 Memory Dump - F5

The F5 (Dump) key is used only in those windows that display values from registers or memory. The dump key uses the value at the current cursor position (the contents of the current register or memory location) indirectly as an address from which to dump memory into the display window. For convenience, the current value, which is the *dump address*, is displayed on the status line along with the current position.

For example, suppose you are in the expression window, the cursor is at an expression *expr*, and the displayed value is `01000h`. If you press F5, the debugger will dump memory starting at address 1000h into the display window.

7.1.4 Disassemble - F6

The F6 (Disassemble) key operates similarly to the F5 key. Instead of dumping memory using the current value, F6 invokes the disassembler. The disassembler uses the current value as an address and disassembles memory into the code window starting at that address. This is useful if you are inspecting an item that represents a label in your code. You can see the code at that label simply by pressing the F6 function key.

7.1.5 Expand the Window - F8

The F8 (Expand) key is used only in the display window. When F8 is pressed, the window expands so that it uses the whole screen. This allows a more efficient view of the file. While the window is expanded, F8 becomes a *Restore* key. Pressing F8 a second time causes the window to be restored to its normal size and the other windows to reappear.

An expanded window is automatically restored to its normal size when you leave it and re-expanded when you return to it.

7.1.6 Update the Screen - F9

The F9 (Update) key forces the debugger to update all displayed values from the emulator and redisplay them on the screen. This is useful after an operation has been performed that may have caused displayed values to change and you wish to see exactly what changed as a result of the operation. The process by which the debugger updates the screen is described in Section 4.7, Updating the Screen.

7.1.7 Online Help - F10

Pressing the F10 (Help) key invokes the debugger's on-line help utility. When help is invoked, a temporary window opens on the screen and help text for the current window is displayed. Additional layers of help can be invoked by following the instructions given in the help text. When help is exited, the temporary window is removed and operation proceeds normally.

7.2 Inspect Code

Table 7-3 describes the commands, functions, and control keys for inspect mode in the code window.

Table 7-3. Inspect Mode Commands Specific to the Code Window

Command	Function
A (Address)	Invoke disassembler at specified address
B (Breakpoint)	Toggle a simple breakpoint at current address
E (Edit)	Invoke the patch assembler at current address
P (PC)	Invoke disassembler at current PC
S (Set PC)	Set PC to current address
X (eXecute)	Run or single-step
F1 (Next)	Move to display window
F2 (Previous)	Move to expression window
F3 (Down)	Scroll forward through code
F4 (Up)	Scroll back through code
F9 (Update)	Update the screen
F10 (Help)	Help for inspect code
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move cursor left one space
R ARROW	Move cursor right one space
UP ARROW	Move up one line, scrolling if necessary
DN ARROW	Move down one line, scrolling if necessary.

In the code window, it is not possible to scroll up past the lowest address that was used when the virtual buffer was first filled with disassembled code. This is because the debugger cannot disassemble backwards through memory. So you can start at a particular address, scroll down, and then scroll back to that address, but no further. If you scroll down past the limit of the buffer, the buffer wraps around and you will not be able to scroll back to the original starting address. You can use the **A**(ddress) command in the code window to get around this problem by invoking the disassembler at any specified address.

While inspecting the code window, the status line displays the address or label at the current cursor position. Example:

```
Inspect Code: [ 701F]
```

7.2.1 Position at a Specified Address - The A Command

Typing **A** (Address) in the code window allows you to move the window to a specified address. The debugger prompts for the address, using the address currently at the top of the window as a default.

7.2.2 Toggle a Simple Breakpoint - The B Command

You can set a simple breakpoint on the instruction at the current cursor position using the **B** (Breakpoint) command while inspecting code. Simple breakpoints are described in the section on Simple Breakpoints on page 5-9. If you type **B** while on an instruction that has no breakpoint set, the debugger sets a simple breakpoint there and displays the reference ID number of the breakpoint to the left of the instruction. If the instruction already has a breakpoint set, the **B** command removes it and the ID number is freed.

7.2.3 Execute Code - The X Command

You can use the **X** (eXecute) command from the code window to execute code without first leaving the code window. The Execute command from the code window operates exactly like the execute command from the top level, described in Section 6, Executing Code. Leaving the execute command causes the debugger to return to the top level and not back to the code window.

7.2.4 Move to the Current PC - The P Command

The **P** (PC) command in the code window positions the window so that the instruction at the current PC is displayed at the top of the window.

7.2.5 Set the PC - The S Command

You can use the **S** (Set PC) command to set the PC to the address of the instruction at the current cursor position. The debugger writes the current address into the PC and highlights the current instruction to show that it corresponds to the PC. This way, you can move to a particular point in code, set the PC there with this command, and begin running from that point with the **X** command.

7.2.6 Edit Code - The E Command

You can use the patch assembler to change the instruction at the current address. After the new instruction is entered the location is modified and the debugger remains in edit mode and points to the next instruction. Either enter <CR> to leave the instruction unchanged and go to the next instruction, enter the new instruction in its place, or enter ESC to exit edit mode.

Relative addressing modes are handled as either an absolute destination address or a relative offset value. If a relative offset is desired, use the correct value. If an absolute address is desired, use a colon prefix (:) before the address. See the examples below:

```
JMP     020h            ;jump 20 bytes ahead
JMP     :7020h          ;jump to location 7020h (must be
```

```
                          ; in range)
BTJO    R1,R2,:7020       ;jump to location 7020h (must be
                          ; in range)
BR      7020h             ;go to location 7020h
BR      :7020h            ;colon makes no difference for absolute
                          ; modes
```

The patch assembler can use all labels and values from a downloaded program but it does not allow new definitions of values. Most assembler directives are not available but the .BYTE and .WORD directives can be used to give one value per line. The assembler does not allow the cursor to move to previous instructions.

7.3 Inspect Memory Dump

Table 7-4 summarizes the commands, functions, and control keys available for inspecting the display window when a memory dump is displayed. Memory is displayed in a hex format, with eight memory locations displayed per line, and ASCII values on the right. The cursor moves between displayed locations on the screen.

Table 7-4. Inspect Memory Dump Commands

Command	Function
A (Address)	Dump memory at specified address
E (Edit)	Modify displayed values
F1 (Next)	Move to CPU window
F2 (Previous)	Move to code window
F3 (Down)	Scroll forward through memory
F4 (Up)	Scroll back through memory
F5 (Dump)	Dump using current value as address
F6 (Disassemble)	Disassemble using current value
F9 (Update)	Update the screen
F10 (Help)	Help for inspect dump
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move cursor to previous location
R ARROW	Move cursor to next location
UP ARROW	Move up one line, scrolling if necessary
DN ARROW	Move down one line, scrolling if necessary
SPACE	Move cursor to next location
RETURN	Exit from 'Edit'
TAB	Move forward 4 locations
Shift-TAB	Move back 4 locations
CTRL L ARROW	Move to beginning of line
CTRL R ARROW	Move to end of line

When inspecting the dump window, the status line displays the address of the current location and the 16-bit value that is stored there. It is this 16-bit value that is used as the address for the F5 (Dump) and F6 (Disassemble) keys. Example:

```
[101F] = F26E
```

7.3.1 Position at a Specified Address - The A Command

Typing **A** (Address) in the dump window allows you to move the window to a specified address. The debugger prompts for the address, using the value at the current cursor position as a default. Type an expression that represents the address at which you want to dump memory. The window is replaced with a memory dump beginning at the specified address.

7.3.2 Interactively Modifying Values - The E Command

The **E** (Edit) command puts you in a special mode that allows you to overwrite the contents of memory simply by typing new values. When you type **E**, the value at the current location is highlighted to indicate that it can be edited. Type the new value in hex. Use the normal movement keys (arrows, space bar) to write the current location and edit another one. Press <CR> to write the current location and leave edit mode.

When you edit a value, whether you change it or not, the debugger writes the edited value to the emulator (so be careful of side effects). After an edited value is written to the emulator, the debugger re-reads and re-displays the value. Thus, if a value cannot be written (for example if no real memory exists at that address), the value will be shown accurately as unchanged on the screen.

7.4 Inspect Peripheral Registers

Table 7-5 summarizes the commands, functions, and control keys for inspecting the peripheral register file. The P registers are displayed in the display window in a two-column format as a result of the D(isplay) P(regs) command.

Table 7-5. Inspect Peripheral Registers Commands

Command	Function
E (Edit)	Modify displayed values
P (Position)	Move window to specified register
F (Format)	Change format of current value
F1 (Next)	Move to CPU window
F2 (Previous)	Move to code window
F3 (Down)	Scroll forward
F4 (Up)	Scroll back
F5 (Dump)	Dump using current value as address
F6 (Disassemble)	Disassemble using current value
F9 (Update)	Update the screen
F10 (Help)	Help for inspecting P registers
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move cursor back one register
R ARROW	Move cursor to next register
UP ARROW	Move up one line, scrolling if necessary
DN ARROW	Move down one line, scrolling if necessary
SPACE	Move cursor to next register
RETURN	Exit from 'Edit'

The status line for inspecting the peripheral registers window includes the current register and its contents, both in hex as follows:

```
[ P01E ] = 00FF
```

7.4.1 Interactively Modifying Values - The E Command

The **E** (Edit) command puts you in a special mode that allows you to overwrite the contents of peripheral registers simply by typing new values. When you type **E**, the value at the current location is highlighted to indicate that it can be edited. Type the new value as a numeric constant, including radix, prefix or suffix. Use the normal movement keys (arrows, space bar) to write the current register and edit another one. Press <CR> to write the current register and leave edit mode.

When you edit a register, whether you change it or not, the debugger writes the edited value to the emulator (so be careful of side effects). After an edited value is written to the emulator, the debugger re-reads and re-displays the value. Thus, if a register cannot be written (for example if the register is

read-only), the value will be shown accurately as unchanged on the screen even if you edit it.

7.4.2 Position at a Specified Register - The P Command

The **P** (Position) command in the peripheral register window allows you to move the window so that a register you specify is positioned at the top. The debugger prompts for a register number; type a decimal number from 0 to 255 or hexadecimal number from 0 to 0FFh. The window is scrolled and refilled so that the specified register is positioned at the top.

7.4.3 Change the Display Format - The F Command

Use the **F** (Format) command to change the displayed format of the peripheral register at the current position. The default display radix is hexadecimal. When you type **F**, the debugger displays a submenu of radix options. The various options are described in Section 3.5, Displayed Values. Select the format you want for the current register. The debugger will convert the value and display it in the selected radix.

7.5 Inspect File

Table 7-6 outlines the commands, functions, and control keys available when you inspect a file in the display window.

Table 7-6. Inspect File Commands

Command	Function
F (Find)	Find a character string
N (Next)	Find the next occurrence of the string
L (Line No.)	Move to specified line number
T (Top)	Move to top of file
B (Bottom)	Move to end of file
F1 (Next)	Move to CPU window
F2 (Previous)	Move to code window
F3 (Down)	Scroll down through file
F4 (Up)	Scroll up through file
F8 (Expand)	Expand window to full screen
F9 (Update)	Update the screen
F10 (Help)	Help for inspect file
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move cursor left one space, scrolling if necessary
R ARROW	Move cursor right one space, scrolling if necessary
CTRL L ARROW	Move to beginning of line
CTRL R ARROW	Move to end of line
INSERT	Scroll window left
DELETE	Scroll window right
UP ARROW	Move up one line, scrolling if necessary
DN ARROW	Move down one line, scrolling if necessary

For inspect file, the status line displays the name of the file, the current line number, and the total number of lines in the file as shown below.

```
"file.txt" line 113/355
```

7.5.1 Find String - The F Command

The **F** (Find) command allows you to search for a particular string of characters in the file. The debugger prompts you for a character string; type a string up to 32 characters long. The debugger begins at the current position and searches for the string. If the debugger reaches the end of the file without finding the string, it wraps around to the beginning of the file and searches from there to the original cursor position. If the string is not found anywhere, the cursor simply remains in its current position. If the string is found, the window is scrolled, if necessary, and the cursor is moved to the start of the target string.

7.5.2 Find Next - The N Command

The **N** (Next) command searches for the next occurrence of a string that was previously found with the Find command. When you type **N**, the debugger begins one space past the current position and begins searching for the same string as used with the most recent Find command. If no Find command has been used, the debugger searches for a null string which is always found at the current position – the cursor does not move. The search proceeds exactly as in the Find command.

7.5.3 Position at a Line Number - The L Command

The **L** (Line Number) command allows you to move to a specified line number in the file. The debugger prompts you for the line number, with the current line as the default. Enter either an absolute line number or a relative offset from the current line, using '+' and '-' in front of the number to indicate a relative offset. The debugger positions the cursor at the specified line number, scrolling the window if necessay.

7.5.4 Position at Top of File - The T Command

The **T** (Top) command positions the cursor on the first line of the file, scrolling the window if necessary.

7.5.5 Position at Bottom of File - The B Command

The **B** (Bottom) command positions the cursor on the last line of the file, scrolling the window if necessary.

7.6 Inspect Symbols

Symbols are displayed in the display window as "lines" of text strings, in exactly the same way that a file is displayed. Therefore, inspecting symbols is very similar to inspecting a file. Many of the same commands are used. Table 7-7 lists the available functions when inspecting symbols.

Table 7-7. Inspect Symbols Commands

Command	Function
F (Find)	Find a character string
N (Next)	Find the next occurrence of the string
T (Top)	Move to first symbol
B (Bottom)	Move to last symbol
F1 (Next)	Move to CPU window
F2 (Previous)	Move to code window
F3 (Down)	Scroll down
F4 (Up)	Scroll up
F9 (Update)	Update the screen
F10 (Help)	Help for inspect symbols
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move cursor left one space, scrolling if necessary
R ARROW	Move cursor right one space, scrolling if necessary
CTRL L ARROW	Move to beginning of line
CTRL R ARROW	Move to end of line
INSERT	Scroll window left
DELETE	Scroll window right
UP ARROW	Move up one line, scrolling if necessary
DN ARROW	Move down one line, scrolling if necessary

For inspect symbols, the status line displays the template used when the symbols were displayed:

```
Symbols "s*"
```

7.6.1 Find String - The F Command

The **F** (Find) command allows you to search for a particular string of characters in the symbol list. You can use the Find command to find symbol names, addresses, module names, or any string that occurs anywhere in the window. The debugger prompts you for a character string; type a string up to 32 characters long. The debugger begins at the current position and searches for the string. If it reaches the end of the symbols without finding the string, it wraps around to the beginning and searches from there to the original cursor position. If the string is not found anywhere, the cursor simply remains in its current position. If the string is found, the window is scrolled, if necessary, and the cursor is moved to the start of the target string.

7.6.2 Find Next - The N Command

The **N** (Next) command searches for the next occurrence of a string that was previously found with the Find command. When you type **N**, the debugger begins one space past the current position and begins searching for the same string as used with the most recent Find command. If no Find command has been used, the debugger searches for a null string which is always found at the current position - the cursor does not move. The search proceeds exactly as in the Find command.

7.6.3 Position at First Symbol - The T Command

The **T** (Top) command positions the cursor at the first symbol in the virtual buffer, scrolling the window if necessary.

7.6.4 Position at Last Symbol - The B Command

The **B** (Bottom) command positions the cursor at the last symbol in the virtual buffer, scrolling the window if necessary.

7.7 Inspect Modules

Inspecting module names is very much like inspecting symbols. In the module window, the debugger's current module is highlighted. Table 7-8 summarizes the available commands and functions.

Table 7-8. Inspect Modules Commands

Command	Function
F (Find)	Find a character string
N (Next)	Find the next occurrence of the string
T (Top)	Move to first module
B (Bottom)	Move to last module
S (Set Current)	Set the current module
F1 (Next)	Move to CPU registers window
F2 (Previous)	Move to code window
F3 (Down)	Scroll down
F4 (Up)	Scroll up
F9 (Update)	Update the screen
F10 (Help)	Help for inspect modules
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move cursor left one space, scrolling if necessary
R ARROW	Move cursor right one space, scrolling if necessary
CTRL L ARROW	Move to beginning of line
CTRL R ARROW	Move to end of line
INSERT	Scroll window left
DELETE	Scroll window right
UP ARROW	Move up one line, scrolling if necessary
DN ARROW	Move down one line, scrolling if necessary

For inspect modules, the status line simply displays the message:

```
Inspect Modules
```

7.7.1 Find String - The F Command

The **F** (Find) command allows you to search for a particular string of characters in the module list. The debugger prompts you for a character string; type a string up to 32 characters long. The debugger begins at the current position and searches for the string. If it reaches the end of the module list without finding the string, it wraps around to the beginning and searches from there to the original cursor position. If the string is not found anywhere, the cursor simply remains in its current position. If the string is found, the window is scrolled, if necessary, and the cursor is moved to the start of the target string.

7.7.2 Find Next - The N Command

The **N** (Next) command searches for the next occurrence of a string that was previously found with the Find command. When you type **N**, the debugger begins one space past the current position and begins searching for the same string as used with the most recent Find command. If no Find command has been used, the debugger searches for a null string which is always found at the current position – the cursor does not move. The search proceeds exactly as in the Find command.

7.7.3 Position at First Module - The T Command

The **T** (Top) command positions the cursor at the first module in the virtual buffer, scrolling the window if necessary.

7.7.4 Position at Last Module - The B Command

The **B** (Bottom) command positions the cursor at the last module in the virtual buffer, scrolling the window if necessary.

7.7.5 Set the Current Module - The S Command

The **S** (Set Current) command sets the debugger's current module to be the one at the cursor position. The current module is the one used to look up local symbols in the symbol table; see Section 5.10, Setting the Current Module – The O Command. The module selected is highlighted in the window to show that it is now the current module.

7.8 Inspect CPU Registers

Table 7-9 summarizes the commands, functions, and control keys for inspecting the CPU register window. The five registers - A, B, PC, SP, and ST - are displayed in fixed positions in the window. The status register is displayed both in hex and in binary.

Table 7-9. Inspect CPU Registers

Command	Function
E (Edit)	Modify displayed values
F1 (Next)	Move to register file window
F2 (Previous)	Move to display window
F5 (Dump)	Dump using current value as address
F6 (Disassemble)	Disassemble using current value
F9 (Update)	Update the screen
F10 (Help)	Help for inspecting CPU registers
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move cursor back one register
R ARROW	Move cursor to next register
UP ARROW	Move up one line
DN ARROW	Move down one line
SPACE	Move cursor to next register
RETURN	Exit from 'Edit'

The status line for inspecting the CPU registers window includes the current register and its contents as follows:

```
[PC] = 702F
```

7.8.1 Interactively Modifying Values - The E Command

The **E** (Edit) command puts you in a special mode that allows you to overwrite the contents of registers simply by typing new values. When you type **E**, the value at the current location is highlighted to indicate that you can edit it. Type the new value in hex, with no prefix or suffix. Use the normal movement keys (arrows, space bar) to write the current register and edit another one. Press <CR> to write the current register and leave edit mode. When you edit a register, whether you change it or not, the debugger writes the edited value to the emulator (so be careful of side effects). After an edited value is written to the emulator, the debugger re-reads and re-displays the value.

7.9 Inspect Register File

Table 7-10 summarizes the commands, functions, and control keys for inspecting the register file. The registers are displayed in a two-column format, all in hex.

Table 7-10. Inspect Register File Commands

Command	Function
E (Edit)	Modify displayed values
P (Position)	Move window to specified register
F1 (Next)	Move to stack window
F2 (Previous)	Move to CPU register window
F3 (Down)	Scroll forward
F4 (Up)	Scroll back
F5 (Dump)	Dump using current value as address
F6 (Disassemble)	Disassemble using current value
F9 (Update)	Update the screen
F10 (Help)	Help for inspecting register file
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move cursor back one register
R ARROW	Move cursor to next register
UP ARROW	Move up one line, scrolling if necessary
DN ARROW	Move down one line, scrolling if necessary
SPACE	Move cursor to next register
RETURN	Exit from 'Edit'

The status line for inspecting the register file window includes the current register and its contents as follows:

```
[R01A] = 00FF
```

7.9.1 Interactively Modifying Values - The E Command

The **E** (Edit) command puts you in a special mode that allows you to overwrite the contents of registers simply by typing new values. When you type 'E', the value at the current location is highlighted to indicate that it can be edited. Type the new value as a hex constant, with no prefix or suffix. Use the normal movement keys (arrows, space bar) to write the current register and edit another one. Press <CR> to write the current register and leave edit mode. When you edit a register, whether you change it or not, the debugger writes the edited value to the emulator (so be careful of side effects). After an edited value is written to the emulator, the debugger re-reads and re-displays the value.

7.9.2 Position at a Specified Register - The P Command

The **P** (Position) command in the register file window allows you to move the window so that a register you specify is positioned at the top. The debugger prompts for a register number; type a decimal number from 0 to 255 or a hexadecimal number from 0 to 0FFh. The window is scrolled or refilled so that the specified register is positioned at the top.

7.10 Inspect Stack

Table 7-11 summarizes the commands, functions, and control keys for inspecting the stack window. The TMS370 has an 8-bit stack pointer, so the stack is located within the register file in the address range 0-255. Each value in the stack window is displayed with its offset from the top of the stack, the absolute address in hex, and the actual contents of the register in hex. For example, a displayed value may appear as:

```
-2(F7) A5
```

meaning that location 0F7h (which is also R247) is two locations below the top of stack and contains the value 0A5h. The location at the top of the stack is displayed with an offset of "SP" and highlighted.

> **Note:**
>
> The stack grows upward (toward larger memory addresses) and the SP points to the item that was pushed onto the stack.

Table 7-11. Inspect Stack Commands

Command	Function
E (Edit)	Modify displayed values
T (Top)	Position window at top of stack
F1 (Next)	Move to expression window
F2 (Previous)	Move to register file window
F3 (Down)	Scroll forward
F4 (Up)	Scroll back
F5 (Dump)	Dump using current value as address
F6 (Disassemble)	Disassemble using current value
F9 (Update)	Update the screen
F10 (Help)	Help for inspecting stack
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move cursor back one register
R ARROW	Move cursor to next register
UP ARROW	Move up one line, scrolling if necessary
DN ARROW	Move down one line, scrolling if necessary
SPACE	Move cursor to next register
RETURN	Exit from 'Edit'

The status line for inspecting the stack window includes the current register and its contents as follows:

```
[-2(F7) ] = 00A5
```

7.10.1 Interactively Modifying Values - The E Command

The **E** (Edit) command puts you in a special mode that allows you to overwrite the contents of registers simply by typing new values. When you type **E**, the value at the current location is highlighted to indicate that it can be edited. Type the new value as a numeric constant, with no prefix or suffix. Use the normal movement keys (arrows, space bar) to write the current register and edit another one. Press <CR> to write the current register and leave edit mode. When you edit a register, whether you change it or not, the debugger writes the edited value to the emulator (so be careful of side effects). After an edited value is written to the emulator, the debugger re-reads and re-displays the value.

7.10.2 Position at the Top of Stack - The T Command

The **T** (Top) command in the stack window scrolls the window so that the register currently pointed to by the stack pointer is positioned at the top of the window.

7.11 Inspect Expressions

Use the expression window to display values that are important to keep track of as you debug your program. You can display any expressions in the window. The debugger evaluates the expression and displays it in the window in whatever format is specified. Each time the screen is updated, the debugger re-evaluates all the expressions in the window and displays the new values, highlighting the ones that have changed. Up to 128 expressions can be stored in the expression window, all of which can be scrolled through.

Table 7-12 summarizes the commands and functions you can use while inspecting the expression window.

Table 7-12. Inspect Expressions Commands

Command	Function
A (Add)	Add a new expression to window
D (Delete)	Remove an expression
E (Edit)	Change the expression
F (Format)	Change the display format of an expression
I (size)	Change data size
N (Name)	Change the label displayed with an expression
L (Load)	Load expressions from a file
S (Save)	Save expressions to a file
F1 (Next)	Move to code window
F2 (Previous)	Move to stack window
F3 (Down)	Scroll forward through expressions
F4 (Up)	Scroll back through expressions
F5 (Dump)	Dump using current value as address
F6 (Disassemble)	Disassemble using current value
F9 (Update)	Update the screen
F10 (Help)	Help for expression window
ESC (Escape)	Leave inspect mode and return to top level command menu
L ARROW	Move up one line
R ARROW	Move down one line
UP ARROW	Move up one line
DN ARROW	Move down one line
SPACE	Move down one line

The status line for inspecting expressions includes the current expression and its value in hex as follows:

```
"expr1" = 01F4
```

7.11.1 Adding a New Expression - The A Command

The **A** (Add) command lets you add a new expression to the window. When you type **A**, the debugger prompts for an expression. Type any valid expression that can be evaluated by the debugger: a symbol name, a register, a memory location, a complex expression using operators, etc.

See Section 3.4, Expressions, for a description of what constitutes a valid expression. Also refer to Section 5.6 for a description of the process of expression evaluation.

The debugger then prompts for the "display name" of the expression. This is the string that is to be displayed next to the expression's value in the window. The default for the name is simply the expression itself. For example, the expression "@SP" represents the 16-bit value on the top of the stack. Assume the expression evaluates to 7006h. You could enter "TOP" as the name for the expression, and the expression would be displayed as:

```
TOP  07006h
```

Note that when this line is displayed on the status line the original expression, not the name, will be shown there:

```
"@SP"  07006h
```

The debugger then displays a submenu that lets you select the radix that will be used to display the value of the expression. The options on this menu are described in Section 3.5, Displayed Values. The default option is 'X' (for hex).

After the radix is selected, the debugger displays another submenu on the same line for you to select the data size of the value. The menu has only two choices: 'B' (Byte) and 'W' (Word). The default is Byte. If you choose Byte, the expression is displayed as an 8-bit value. If you choose Word, the expression is displayed as a 16-bit value. All internal computations are done with words so beware of side effects particularly when using the shift right operator.

After the above four items have been specified, the debugger inserts the new expression into the window just after the expression at the current position. The expression remains in the window until you either remove it with the Delete command or load an object file, in which case all expressions are removed.

7.11.2 Deleting an Expression - The D Command

When the **D** (Delete) command is typed in the expression window, the expression at the current cursor position is removed from the window. Other expressions below the deleted one are scrolled up to fill the vacated line in the window.

7.11.3 Editing an Expression - The E Command

Use the E (Edit) command to change the expression. When you type E, the debugger displays the expression on the command line for editing.

7.11.4 Change the Display Format - The F Command

Use the **F** (Format) command to change the displayed format of the expression at the current position. When you type **F**, the debugger displays the same submenu of radix options that is used for adding expressions. The various options are described in Section 3.5, Displayed Values. Select the format you want for the current expression. The debugger will convert the value and display it in the selected radix.

7.11.5 Change the Data Size - The I Command

The I (size) command allows changing the displayed size from byte to word or vice versa. While the displayed size of an expression can be changed, all internal computations are done using 16 bits. Beware of any side effects particularly when using the shift right operator.

7.11.6 Change the Display Name - The N Command

The **N** (Name) command allows editing of the expression name that appears with the current expression (at the left of its value). When you type **N**, the debugger displays the name on the command line for editing.

7.11.7 Load Expressions From a File - The L Command

The **L** (Load) command is used to add expressions saved in a file to the window. Expressions can be saved with the Save command, described in the following section. When **L** is typed, the debugger prompts for a filename from which expressions are to be read. You can omit the extension on the file name; the default extension is `.exp`. The debugger reads all the expressions from the file, evaluates them, and inserts them into the window. Any previous expressions remain in the window.

The expression file contains the name, expression, radix, size, and module number for each expression. Be sure that any symbols that are used in saved expressions are available to the debugger when the saved expressions are reloaded. For example, assume you save an expression window while debugging an object file called `prog.out` and that expressions in the window contain symbols from that object file. If you want to load the saved expression file in a subsequent debugging session, you must first load `prog.out` so that the symbols used in the saved expressions are available to the debugger.

If any expression in the file cannot be evaluated when the file is loaded, an error message is displayed and the expression is ignored. The debugger proceeds with reading the remaining expressions from the file.

7.11.8 Save Expressions to a File - The S Command

Use the **S** (Save) command to build a file of expressions so that they can be reloaded at a later time. When **S** is typed, the debugger prompts for the name of the file into which the expressions are to be saved. You can omit the extension on the filename; the default extension is `.exp`. If the file exists, it will be replaced with the new expressions. Each expression in the virtual buffer (not just the ones currently on the screen) is written to the file. The file contains the name, expression itself, radix, size, and module number for each expression.

Section 8

Hardware Breakpoints, Tracing and Timing

The TMS370 emulator contains a separate board that provides circuitry to monitor the operation of the system at a hardware level. This board is called the BTT (Breakpoint, Trace, and Timing) board. The BTT board operates by monitoring the CPU as it runs and taking certain actions when particular patterns of bus activity are detected. You specify a set of qualifiers that define the bus cycle patterns being looked for, as well as the actions to be taken when the qualifiers are matched. This specification process is called **programming** the BTT. Each action has an associated set of qualifiers. Refer to Figure 8-1 and Figure 8-2 (on page 8-10) while reading this section.

This section includes the following topics:

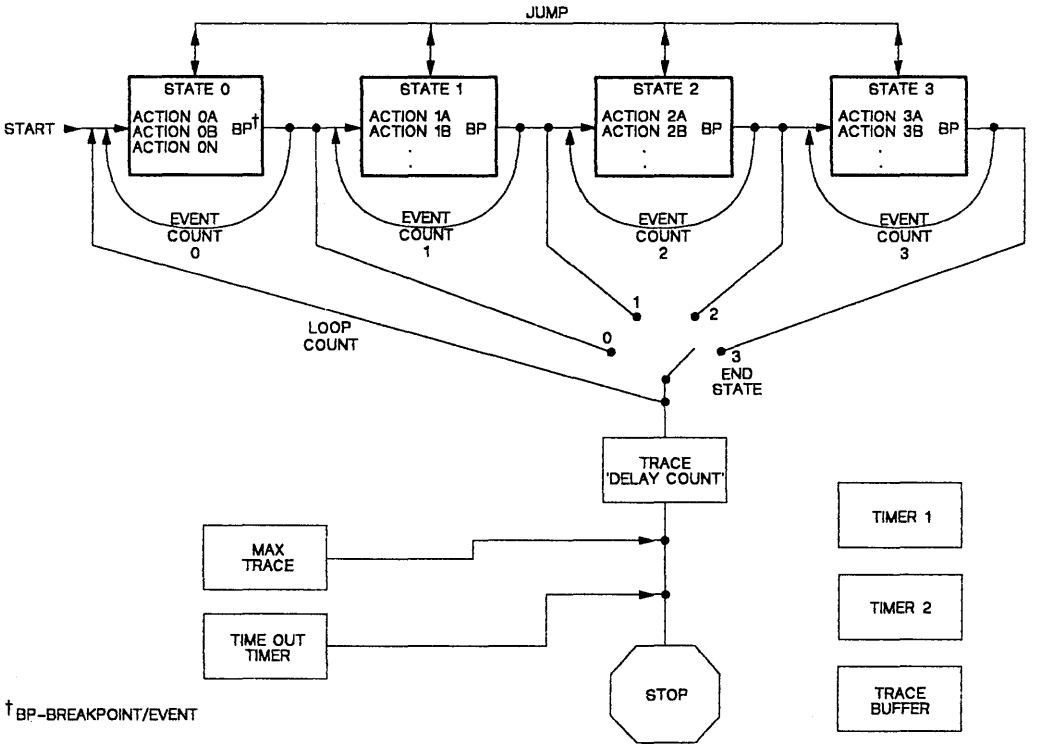


Figure 8-1. Understanding the BTT

8.1 Actions

The various actions that the BTT can perform when a set of qualifiers is matched are described below.

BP/EVENT Triggering a breakpoint/event may either cause a hardware breakpoint, decrement the event counter, or transfer control to the next state. The relationship between events and breakpoints is described in the section on Events and Breakpoints on page 8-7.

TRACE A cycle which satisfies the TRACE qualifiers will be stored in the TRACE buffer. This provides a history of program execution for later inspection.

JUMP The BTT has four separate states in which different sets of actions can be specified. The JUMP action forces a transition into a different state when triggered.

POINT TIMER The BTT has two timers that can be started or stopped by qualified actions. The POINT TIMER action uses the two address qualifiers to control one timer. The timer is started when the first address is qualified and stopped when the other address is qualified.

RANGE TIMER The RANGE TIMER actions also control the BTT timers but differ from the point timer action in that one action starts a timer and a separate action stops it. Thus, there are actually two actions, *range timer start* and *range timer stop*.

8.2 Qualifiers

The qualifiers that can be monitored by the BTT to trigger any one action are described below. An action is triggered when a cycle occurs that matches all the qualifiers for the action.

ADDRESS The BTT monitors the memory bus during all memory cycles. Two address qualifiers can be used to trigger an action on a particular address or range of addresses. These can be used to define two distinct single point addresses, an inclusive range (any address within the range qualifiers), or an exclusive range (any address outside the range qualifiers). You can specify a mask to selectively ignore some or all of the address qualifier bits.

DATA The BTT also monitors the value on the data bus during each memory cycle. Two data qualifiers can be used with the data bus in exactly the same way as the two address qualifiers are used with the address bus. You can specify a mask to selectively ignore some or all of the data qualifier bits.

CYCLE You can specify which types of memory cycles qualify to trigger an action. Memory cycle types are read, write, and instruction fetch. Either a single type or combinations of cycles can be qualified.

EXTERNAL The BTT monitors the logic level of the eight external probe lines. Refer to Section 11.4.3, page 11-18, for more specific information about the probes. There is a qualifier that can be used to trigger an action on a particular value from these inputs. You can specify a mask to selectively ignore some or all of the external qualifier bits. Refer to Section 8.8.2 and Section 8.8.9 for information on masks and editing qualifiers.

8.3 States

The BTT has four independent sets of actions available. Each set of actions is called a **state**. Up to four actions can be specified per state. When the BTT is running, it is always in one of the four states. Only the actions in the current state can be triggered; the other states are inactive. Qualifiers for actions in an inactive state are ignored until the BTT makes a transition into that state.

All the actions programmed in the current state are simultaneously active. In other words, any cycle can trigger any or all of the actions in the state.

The states are numbered 0–3. When the BTT is reset, it always begins in state 0. The BTT can transition from one state to another in two different ways. First, a JUMP action can be programmed in a state. If the JUMP action is triggered, the BTT will change to a specified destination state. Second, the BTT can automatically change states when a certain number of EVENT actions occur. The conditions under which this occurs are described in the section on Events and Breakpoints on page 8-7.

Each state can be programmed in **address only** or **address+data** mode. In address-only mode, the data qualifiers are unavailable, and you can have up to four actions in the state. In address+data mode, the data qualifiers are available, but the state can contain a maximum of only two actions. The number of actions in each state is also subject to various limitations that are described in the section on Resource Allocation.

In addition to the mode, each state has additional parameters and settings that determine how the board acts in that state. Different states can have different values for these parameters. These are referred to as **local** settings since they affect only one state. The various local settings are described in Section 8.9, Local State Settings. In addition, there are a number of parameters that affect the operation of the board in all states. There is only one set of these parameters, so they are called **global** settings (see Global Settings on page 8-27).

8.3.1 Resource Allocation

Not all combinations of the maximum four actions (in address-only mode) or two actions (in address+data mode) can always be programmed in a state. Due to limitations of the BTT hardware, there are a fixed number of possible configurations for each state. Each possible configuration represents a combination of actions that can be used. Other combinations of actions are not allowed.

The debugger keeps track of what actions have been used and what are available as you program a state. It displays a summary of the available actions based on the possible combinations. In this way, allocating actions into a state such that they form a valid combination is somewhat transparent. The debugger prevents you from specifying an action that would result in an invalid combination and displays an error message if you try.

Table 8-1 and Table 8-2 show all the possible configurations of actions that can be allocated into any one state for both address+data mode and address-only mode, respectively. The numbers in the tables represent how many times an action can be used. A blank entry means that no actions of that type can be used.

For example, in address+data mode, it is possible to have one EVENT and one JUMP in a state (from line 3 of Table 8-2). Or, you can have one TRACE and one POINT TIMER (line 6). However, it is not possible to have a TRACE and a JUMP in the same state.

In address-only mode, you can have two EVENTs, a TRACE, and a JUMP (from line 9 of the second table). However, it is not possible to have three JUMPs in any state.

Table 8-1. Address-Only Configurations

Number	BP/ Event	Trace	Jump	Point Timer	Range Timer
(1)	4				
(2)	3	1			
(3)	3		1		
(4)	3			1	
(5)	2	2			
(6)	2		2		
(7)	2				1
(8)	2			2	
(9)	2	1	1		
(10)	2	1		1	
(11)	2		1	1	
(12)	1	3			
(13)	1	2	1		
(14)	1	2		1	
(15)	1	1	2		
(16)	1	1			1
(17)	1		2	1	
(18)	1		1		1
(19)	1		1	2	
(20)	1			1	1
(21)		4			
(22)		2			1
(23)					2

Numbers in the left column refer only to this table.

Table 8-2. Address + Data Configurations

Number	BP/ Event	Trace	Jump	Point Timer	Range Timer
(1)	2				
(2)	1	1			
(3)	1		1		
(4)	1			1	
(5)		2			
(6)		1		1	
(7)					1
(8)				2	

Numbers in the left column refer only to this table.

8.3.2 Events and Breakpoints

EVENT actions in the BTT can be used to cause a hardware breakpoint that halts the CPU and BTT. (The CPU is halted only in normal run mode.) Various counters on the BTT, along with the state transition scheme, determine when an actual hardware breakpoint is caused. The counters start at a value you specify when programming the BTT and count towards 0.

Each state has an EVENT counter. The initial value of the event counter is one of the local state settings. Each time an EVENT occurs in the state, the event counter is decremented. When an EVENT occurs that decrements the event counter to zero, the BTT automatically sequences to the next higher state. The BTT can also change states with JUMP actions.

One of the global settings on the BTT board is an **end state**. If the event counter in the end state reaches zero, the BTT has completed its state sequence. At this point, another counter called the **loop counter** is decremented. The initial value of the loop counter is another global setting. If the loop counter is not 0 after the completion of the entire state sequence (when the event counter in the end state reaches 0), the BTT resets to state 0 and begins another state sequence.

When the loop counter reaches 0, another counter, called the **delay counter**, begins to take effect. The delay counter is decremented (after the loop counter expires) each time a TRACE action occurs using the end state's qualifiers. The initial value of the delay counter is another global setting. When the delay counter reaches 0, a hardware breakpoint occurs. The delay counter allows you to take a number of trace samples before halting, so that you can see what happens just after an event occurs.

In summary: The BTT sequences through a cycle of states, each of which has an associated event counter. When the event counter in a state expires (reaches zero), the BTT sequences to the next state. The BTT completes a cycle when the event counter in the end state expires; the loop counter is then decremented and the BTT begins again at state 0. When the loop counter expires, the BTT waits for a number of trace samples (specified by the delay count) to be taken. When the delay count expires, a hardware breakpoint

occurs and the BTT halts. The flowchart in Figure 8-2 illustrates the logical operation of the BTT.

8.3.3 Timers

The BTT has two timers that can be programmed to start and stop by qualified actions. The timers can be used to do timing analysis on your program in real time as the program runs. There are two actions used to control these timers: POINT TIMER and RANGE TIMER. Note that these two different actions share the same two timers; the two actions are simply different ways of starting and stopping the timers.

The timers are like stopwatches. Both timers start at 0 when the BTT is reset. When a POINT TIMER or RANGE TIMER action that starts a timer is qualified, the timer starts and begins accumulating elapsed time. The timer continues to run until a POINT TIMER or RANGE timer action to stop the timer is qualified. Like a stopwatch, the timer can be started and stopped many times; when a timer is re-started it does not reset to 0 but continues to accumulate time from the point at which it was stopped.

The timers are designated **timer #1** and **timer #2**. Both timers behave the same, except that timer #1 has an *averaging* feature that timer #2 does not have. Timer #1, in addition to accumulating elapsed time, keeps track of the number of times it is started. Then, when the program finishes running, timer #1 calculates the average amount of time that elapsed each time the timer ran. Average time is calculated by dividing the total accumulated time by the number of times the timer was started. For example, suppose that after the program begins running, timer #1 is started and then stopped 7 seconds later. Then, timer #1 is started again and stopped 3 seconds later. The total accumulated time for timer #1 is 10 seconds (7+3). The average time is 5 seconds (10 seconds total divided by 2 starts).

The values of the two timers, and the average value for timer #1, are displayed on the trace screen of the debugger. See Section 5.8, Inspect Trace Samples - The T Command.

A single POINT TIMER action uses its two address qualifiers (all actions have two address qualifiers) to start and stop the timer. The first address starts the timer and the second stops it. The other qualifiers are shared between the two addresses and apply to both starting and stopping the timers. For example, suppose a POINT TIMER action is specified as follows:

```
ACTION: PointTimer
timer #1
addr1 = 7000h
addr2 = 7FF0h
data  = IGNORE
 cycles IAQ
 extern IGNORE
```

When an IAQ at address 7000h occurs, timer #1 is started. The timer runs until an IAQ at 7FF0h occurs, at which point it is stopped.

A RANGE TIMER action, on the other hand, uses two separate actions: one dedicated to starting the timer and another to stopping it. These two actions

are completely independent. For example, suppose a RANGE TIMER action is specified as follows:

```
ACTION: RangeTimer
start #2
addr1 = 7000h
addr2 = 7FF0h
data = IGNORE
cycles IAQ
extern IGNORE
```

```
ACTION: RangeTimer
stop #2
addr = IGNORE
data = 0FFh
 cycles MR
 extern IGNORE
```

When an IAQ from address 7000h or 7FF0h occurs, timer #2 is started. This is controlled by the first action. The timer stops when a memory read cycle reads the data value 0FFh from any address in memory. This is controlled by the second action.

The first timer action in a state always uses timer #1. The second timer action uses timer #2. If you have two timer actions and delete the first one, the second timer action switches from using timer #2 to timer #1 (since it is now the first timer action). Which timer is being used by an action is displayed under the action type field in the list of qualifiers.

You can control a timer from two different states. If a timer is started in one state and the BTT sequences to a different state, an action in the new state can stop the timer. Timers are not affected by transitions between states.

The BTT has an additional time-out timer that can be programmed to stop your program when a specified amount of time has passed. This time-out timer is independent of the two programmable timers. You specify a time-out value in the BTT global settings window, described in Section 8.10, Global Settings.

8.3.4 Trace Buffer

The BTT Trace Buffer is a circular buffer with a capacity of 2047 trace samples. If more than this number of samples is collected, the buffer wraps around and overwrites old samples. Trace samples are also discussed in Trace Samples on page 5-14.

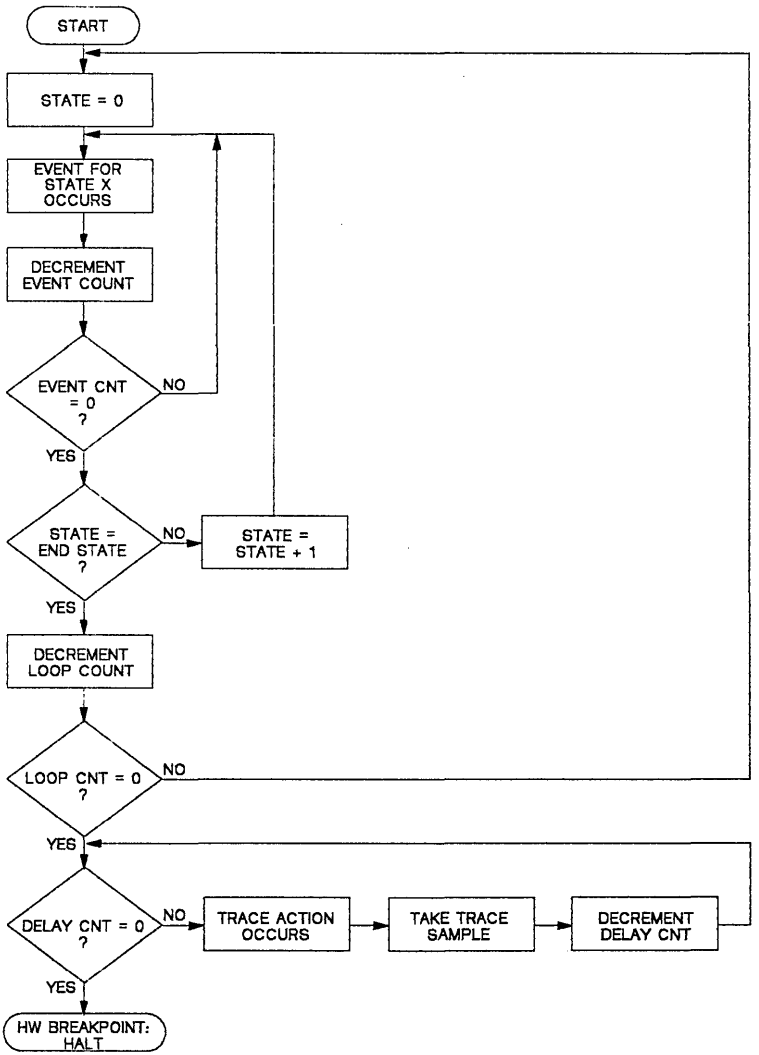


Figure 8-2. Flow of BTT Operation

8.4 Sample BTT Programs

8.4.1 A Simple Example

Few situations require multiple states or use of all counters. A common case is to set up the following situation using default values for the counters:

- Set a BP/Event action in state 0
- Set Event Count = 1 (default)
- Set End State = 0 (default)
- Set Loop Count = 1 (default)
- Set Delay Count = 0 (default)

When the BP/Event is triggered: the event count is decremented and expires immediately. No state transition occurs, because the end state is 0. The loop count is decremented and expires immediately. The delay count is already 0, so it also expires immediately. Thus, a hardware breakpoint occurs as soon as the EVENT action is qualified.

8.4.2 Using the Event Counter

For the second example, assume that the program to be debugged has a long software loop. Initialization occurs before entering the loop and then the loop is repeated several times. The program appears to operate correctly the first time through the loop but fails on each succeeding pass.

To catch a trace sample the second time through the loop, use the event counter.

1) Set a BP/Event in state 0 to occur when the address is equal to the beginning of the loop during an IAQ cycle.

2) Set a Trace action in state 0.

3) Set the Event Count to 2.

4) Leave the End State at 0.

5) Leave the Loop Count at 1.

6) Set the Delay Count to 1000.

This setting causes the emulator to continuously capture trace samples. When the trace buffer is full, the oldest sample is "pushed out" to make room for a new sample. When the program fetches the instruction at the top of the loop for the second time, the emulator starts counting and allows 1,000 more memory cycles to be captured in the trace buffer before it stops execution.

The end result is that the trace buffer contains a record of the last 1047 memory cycles from the first time through the loop, and the 1,000 memory cycles from the second time through the loop.

8.4.3 Using Multiple States

For the third example, assume that your program calls a subroutine to read a character from another machine. This subroutine is called from several locations in the program, but you only get unexpected results when it is called from one specific location.

Setting a BP/Event at the point that the routine is called will not guarantee that the subroutine can be traced, since the routine waits an indefinite time for the character to come from the other machine. Setting the BP/Event after a character is received may cause the emulator to trace when the subroutine is called from some other part of the program.

The solution is to use multiple states. In state 0, set a BP/Event to occur on the instruction acquisition of the call to the receive character routine. In state 1, set a BP/Event to occur when a character is actually detected. Also set a Trace in state 1. Now set the delay count so that you can see a few cycles before the acquisition of the character and several cycles following the acquisition.

1) Set a BP/event in state 0 to occur when the address is equal to the subroutine call during an IAQ cycle.

2) Set a BP/event in state 1 to occur when the address is equal to the instruction following the successful detection of a character.

3) Set a trace action in state 1.

4) Leave the event count in both states at 1.

5) Set the End State to 1.

6) Leave the Loop Count at 1.

7) Set the Delay Count to 1000.

8.4.4 Using The Loop Counter

For the fourth example, assume that you need to measure the average time our subroutine spends waiting for a character when it is called from a specific part of the program. When you need to repeat sequences that require the use of more than one state, you use the loop counter.

1) Set a BP/Event in state 0 to occur when the address is equal to the subroutine call during an IAQ cycle.

2) Set a Range Timer in state 1 to start when the address is equal to the instruction where the program begins checking for reception of a character.

3) Set the Range Timer in state 1 to stop when the address is equal to the instruction where the program has detected the reception of a character.

4) Set a BP/Event in state 1 to occur when the address is equal to the instruction where the program has detected the reception of a character.

5) Leave the event count in both states at 1.

6) Set the End State to 1.

7) Set the Loop Count to 1000.

When the program halts (due to a hardware breakpoint), the trace buffer display will show the total time spent waiting for 1,000 characters to arrive in the Timer 1 register. The average time spent waiting for a character will be shown in Timer 1 Average.

8.5 Programming the BTT

The debugger keeps a representation of the current configuration of the entire BTT board that includes all the actions, qualifiers, local state settings, and global settings. You can use the special BTT screen of the debugger to display and modify any of the parameters. When you exit the BTT screen, the debugger programs the BTT with the configuration you have built.

To get to the BTT screen from the top level command menu of the debugger, type **BB**. The first **B** selects the breakpoint command from the top level and the second selects the BTT option of the breakpoint command.

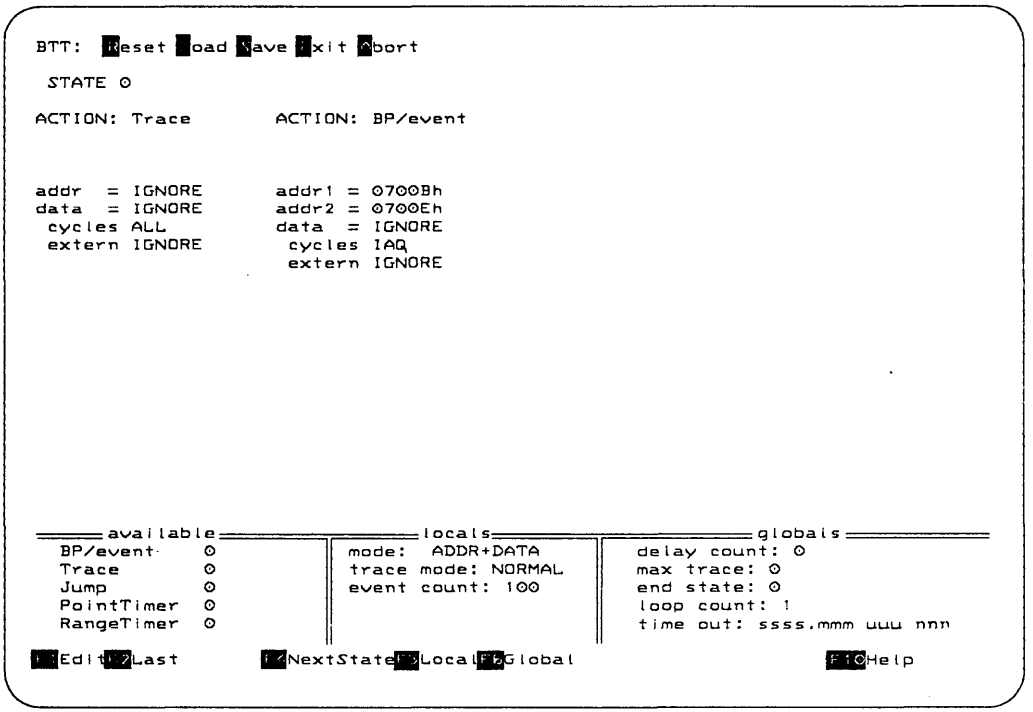


Figure 8-3. BTT Screen

An example of the BTT screen is pictured in Figure 8-3. The screen displays the settings for one state at a time; you change the currently displayed state with function key F4. The currently displayed state is indicated on the third line of the screen, toward the left. The central area of the screen is for the four possible actions in the state and their associated qualifiers. When no actions are programmed into the state, this area is blank. As actions are added to the state, the area fills up with action specifiers and qualifiers. The example in Figure 8-3 shows two of the four action areas filled. The bottom third of the screen is divided into three windows named *available*, *locals*, and *globals*.

The leftmost window indicates what actions are currently available. Each type of action is listed along with a number that indicates how many of that action type remain available for use in the state. When you program an action, the window is updated to reflect the new availability, according to the table of possible combinations and the currently programmed actions.

The middle window shows the local state settings for the currently displayed state. The meaning of each setting and how to change it is described in Section 8.9, Local State Settings. When you change the displayed state, the values in this window change also to reflect the settings for the new state.

The rightmost window shows the global settings for the BTT. The meaning and programming of these settings are described in Section 8.10, Global Settings.

The BTT screen has a command menu like the menus in other parts of the debugger. You can move off the command line into the center area of the screen to enter or modify actions and qualifiers. You can move to the local settings window to change any parameters there, or you can move to the global settings window to change global parameters.

8.6 BTT Command Window

Table 8-3 summarizes the commands and functions available from the BTT command menu. These functions are described in detail in the sections that follow.

Table 8-3. BTT Command Window

Command	Function
R (Reset)	Clear the configuration
L (Load)	Load BTT configuration from file
S (Save)	Save BTT configuration into file
E (Exit)	Exit BTT screen and program BTT with current configuration
A (Abort)	Exit BTT screen but do not program BTT
ESC (Escape)	Exit and program - same as 'E'
F1 (Edit)	Move to the first action area to add, edit, or remove an action
F2 (Last)	Move to the last action area to add, edit, or remove an action
F4 (NextState)	Display the next BTT state
F5 (Locals)	Move to local state settings and edit them
F6 (Globals)	Move to global settings and edit them
F10 (Help)	Help for BTT command menu

8.6.1 Clear BTT Configuration - The R Command

The **R** (Reset) command in the BTT command menu clears the entire BTT configuration. All actions are removed from all states, so that no actions remain programmed. The local state settings for each state and the global settings are reset to their default values. If the BTT is programmed with a cleared configuration, it will perform no operations when it runs.

Using the Reset command does not actually program the BTT with the cleared configuration; this does not occur until you leave the BTT screen. You can use the Reset command to clear an earlier setting and then program new actions and settings from scratch.

8.6.2 Load a BTT Configuration - The L Command

The **L** (Load) command of the BTT menu loads a complete BTT configuration from a file that has been created previously with the **S** (Save) command. The configuration in the file includes actions and qualifiers for all states, local state settings for all states, and global settings. When you type **L**, the debugger prompts for a filename. You can omit the extension on the filename; the default extension is ".btt". If the file exists, the debugger loads it and redraws the screen to reflect the newly loaded configuration. You can then modify the new configuration or use it directly to program the BTT by exiting from the BTT screen.

8.6.3 Save a BTT Configuration - The S Command

The **S** (Save) command of the BTT menu saves the current BTT configuration into a file. This way, you can restore a particular configuration later using the **L** (Load) command. When you type **S**, the debugger prompts for a filename. You can omit the extension on the filename; the default extension is ".btt". If a file of that name exists, it is replaced by the new file. The **S** command does not affect the current configuration in any way.

8.6.4 Exit and Program BTT - The E Command

Use the E (Exit) command of the BTT menu or the <Esc> key to program the BTT with the current configuration and return to the main screen of the debugger. The next time you use the Execute command to run code, the BTT behaves in the way you have specified.

8.6.5 Exit Without Programming - The A Command

The **A** (Abort) command leaves the BTT screen and returns to the main debugging screen without actually programming the BTT. The BTT remains configured as it was the last time it was programmed. The Abort command is useful when you are setting up a BTT configuration and need to temporarily check something on the main screen, such as the location of a particular instruction. You can Abort, check, and return to the BTT screen without programming the board itself.

> **Note:**
>
> The BTT screen will **not** reflect the actual state of the BTT if you Abort; it will contain the last set of your modifications.

8.6.6 Edit Next Action - F1

The F1 (Edit) function key is used to move to the action area of the currently selected state and add a new action, modify an existing action, or remove an action. Typing F1 from the command line moves you to the first (leftmost) action on the screen. Successively typing F1 moves forward through the four action areas and then back to the command line. The editing process for BTT actions is described in Section 8.7, Editing Actions.

8.6.7 Edit Previous Action - F2

The F2 (Last) function key is like the F1 key, except that it moves you from the command line to the last (rightmost) action area. Successively typing F2 moves backward through the four action areas, rather than forward.

8.6.8 Change States - F4

The F4 (NextState) function key cycles through the four states. Use this key to change the displayed state so that you can set up actions or local state settings in different states. When a new state is displayed, the debugger updates the actions, availability window, and local state settings window to reflect the configuration for the new state. The currently displayed state is indicated on the third line of the screen, toward the left.

8.6.9 Edit Local State Settings - F5

The F5 (Local) function key moves to the local state settings window on the bottom of the screen so that you can modify the values there. The meaning and editing of these values is described in Section 8.9, Local State Settings.

8.6.10 Edit Global Settings - F6

The F6 (Global) function key moves to the global settings window on the bottom of the screen so that you can modify the values there. The meaning and editing of these values is described in Section 8.10, Global Settings.

8.7 Editing Actions

When you move to the central area of the screen with the F1 (Edit) key, you can enter or modify the actions that are programmed into the current state. Each of the four "action areas" in the center of the screen consists of a list of settings that specify an action and the qualifiers that are used to trigger the action. The action areas are filled in from left to right as you enter new actions into the state.

Use F1 (Next) and F2 (Prev) to move forward and backward through the actions. Once an action is allocated into a state, the action itself cannot be changed. You can, however, move to the action and change its qualifiers. Or, you can delete the action completely and then enter a different action. For example, suppose you have entered an EVENT action into the state with an address qualifier of 7000h. You cannot change the EVENT to a TRACE, but you can change the address from 7000h to 7010h.

A number of function keys can be used to perform special actions while editing actions and qualifiers. Table 8-4 lists what these keys are and their effect.

Table 8-4. Function Keys for Editing Actions and Qualifiers

Command	Function
F1 (Next)	Move to next action
F2 (Prev)	Move to previous action
F3 (Clear)	Clear the current action
F4 (NextState)	Display the next BTT state
F5 (Local)	Move to local state settings and edit them
F6 (Global)	Move to global settings and edit them
F10 (Help)	Help
ESC (Escape)	Return to BTT command menu
UP ARROW	Move up to previous qualifier
DOWN ARROW	Move down to next qualifier

Most of these keys have already been described. Function keys F1, F2, F4, F5, and F6 operate in exactly the same way as on the command menu and are described in Section 8.6, BTT Command Window.

8.7.1 Entering New Actions

To enter a new action, use F1 (or F2) to step over existing actions to the first blank action area. Initially, all the areas are blank and a new action can be entered by typing F1 just once.

The first setting in each list is for the action type. When you move to an action area, the cursor is positioned at this setting. For a new action, the debugger displays a submenu of options that you use to select the action type, as summarized below:

Table 8-5. New Action Keys

Command	Function
B (BP/Event)	Selects EVENT action
T (Trace)	Selects TRACE action
J (Jump)	Selects JUMP action
P (PointTimer)	Selects POINT TIMER action
R (RangeTimer)	Selects RANGE TIMER action

The various actions are described in Section 8.1, Actions. A new action can be entered into the state by selecting one of the options from the menu. Check the availability window on the bottom of the screen to see if the action is available. If the number corresponding to the action is greater than zero, the action is available. If you select an action that is not available, the debugger displays an error message. Otherwise, the debugger allocates the new action into the state, updates the availablity window, and displays the list of qualifiers for the action, with default values, under the action. RANGE TIMER actions are allocated in pairs: one action to start the timer and a separate one to stop it. Because of this, RANGE TIMER actions fill two of the four areas on the screen when they are allocated.

The cursor is positioned on the first qualifier. You can then enter the qualifiers for triggering the new action. See Section 8.8, Editing Qualifiers.

8.7.2 Removing an Action - F3

An action can be removed from a state by moving to it and pressing F3 (Clear). When an action is removed, the other actions are moved to the left in order to fill the space vacated by the deleted action. The availability window is updated to reflect the fact that one action has just been freed.

Note that RANGE TIMER actions are allocated in pairs; when you delete either the *start* or *stop* action of a range timer, the other action is deleted also.

You can edit the qualifiers of an action by using the DOWN ARROW key to move to the proper position. You can move to another action by pressing F1 (Next) or F2 (Prev). You can move to the local state settings or global settings windows with F5 (Local) or F6 (Global). Or you can return to the command line with the <Esc> key.

8.8 Editing Qualifiers

Qualifiers are listed on the screen below the action they represent. The qualifiers for an action are used to determine under what conditions the action is triggered. The various qualifiers are explained in Section 8.2, Qualifiers. When you initially enter an action, the qualifiers appear as shown below (assuming address+data mode):

```
Action:  BP/Event
addr   = IGNORE
data   = IGNORE
 cycles ALL
 extern IGNORE
```

The arrow keys are used to move up and down through the qualifiers, making any desired changes.

Many of the function and control keys used throughout the BTT screen are also valid when specifying qualifiers. Use of these keys has been explained in previous sections. There are three special keys that are used only with the qualifiers: F7 (Range), F8 (Mask), and F9 (Ignore). A description of these keys follows.

Table 8-6. Function Keys for Editing Qualifiers Only

Command	Function
F7 (Range)	Select address interpretation
F8 (Mask)	Define a bitmask for a qualifier
F9 (Ignore)	Set a qualifier to be ignored (mask 0)

8.8.1 Range Specifications - F7

There are two address and two data qualifiers for each action on the BTT board. Table 8-7 shows the ways that the two address qualifiers can be interpreted and how the qualifiers are displayed for each interpretation. The *qualification* column in the table gives the condition that causes an input value of "x" on the address bus to qualify. The data qualifiers work in exactly the same way.

Table 8-7. Interpreting Range Specifications

Interpretation	Display	Qualification
Single point	addr = <value>	x = <value>
Two points	addr1 = <value1> addr2 = <value2>	x = <value1> OR x = <value2>
Inclusive range	addr = <value1> ... <value2>	x >= <value1> AND x <= <value2>
Exclusive range	addr != <value1> ... <value2>	x < <value1> AND x > <value2>

The default interpretation is *single point* for all qualifiers. When the cursor is positioned on any address qualifier or any data qualifier, use the F7 (Range) key to cycle through the four interpretations shown in Table 8-7.

8.8.2 Masks - F8

The address, data, and external probe qualifiers on the BTT each have a mask associated with them that allows you to selectively ignore bits of the address bus, data bus, or external probe inputs. The address qualifiers have a 16-bit mask, and the others have 8-bit masks. Before a value from the bus (or probes) is compared with the qualifiers, the associated mask is logically ANDed with both the value and the qualifiers. Thus, any zero bits in the mask force the associated bits in both the input value and the qualifier to be zero also, thereby causing those bits to be equal and effectively ignored.

For example, assume it is necessary to qualify an action based on the external probe inputs, but you need only four of the eight bits. You want the action to be triggered when the four bits have a binary value of 1010, or 0Ah. You can use the four least significant probe inputs, select a value of 0Ah for the external qualifier, and a mask of 0Fh. This way, the top four bits are masked off and ignored. When the lower four bits have a value of 0Ah, the qualifier is satisfied.

By default, if a value is specified for a qualifier, all the mask bits are ones so the mask has no effect. In this case, the mask is not displayed on the screen with the other qualifiers. You can bring the mask for a qualifier onto the screen by typing the F8 (Mask) key. This allows you to edit the mask value, changing some of the bits to zero if desired. If you type F8 when a mask for the qualifier is already on the screen, the mask is set to all ones and removed from the screen.

When the mask is enabled, it is displayed in the qualifier list below the associated address, data, or external qualifiers. For example:

Before typing F8:

```
ACTION: TRACE

addr =  7000h        <-- current position
data =  IGNORE
 cycles ALL
 extern 0FFh
```

After typing F8:

```
ACTION: TRACE

addr =  7000h
   mask 0FFFFh       <-- current position
data =  IGNORE
 cycles ALL
 extern 0FFh
```

The desired value for the address mask can now be entered.

A special case occurs when a qualifier is set to **IGNORE**, as described in the next section. In this case, the mask is 0 but not displayed on the screen.

8.8.3 IGNORE a qualifier - F9

The address, data, and external probe qualifiers have a special setting called *IGNORE*, which simply means that the mask is set to 0 for that qualifier. Now ANY value will qualify; the entire qualifier is effectively ignored.

When a new action is entered, all the qualifiers initially default to *IGNORE*. This means that every bus cycle qualifies and will trigger the action. The action can be made more selective by editing only the qualifiers of interest.

A qualifier can be forced to the *IGNORE* setting by typing the F9 (Ignore) key when positioned at the qualifier. This sets the mask to 0 and displays `IGNORE` for the value of the qualifier.

8.8.4 Expressions as Qualifiers

Where qualifiers have numeric values, any expression can be entered. For BTT qualifiers, expessions are limited to 11 characters in length. When an expression is entered it is evaluated immediately and the resulting static value used to program the BTT. If you leave the currently displayed state (either by moving to a different state or exiting the BTT screen) and then return, all settings are displayed with their numeric values only.

For example, assume you enter "PC+SYM" for an address qualifier. The PC is currently 7000h and SYM has a value of 0F3h. The value stored in the qualifier is 70F3h. Now assume you return to the main screen, change the PC to 8000h, and return to the BTT screen. The address qualifier is displayed as `70F3h`, representing the VALUE it was originally programmed with and not the expression used to get that value.

8.8.5 Action Specific Prompts

If you select a JUMP action, the debugger prompts for the jump destination state on the line below the action prompt, as shown below:

```
Action: JUMP
        to ___
```

Enter the state (0–3) that you want the BTT to enter if the JUMP action is qualified.

8.8.6 Address Prompts

The address qualifiers are displayed at the top of the qualifier list. There are either one or two addresses (depending on the range interpretation) and a mask (if enabled). For each prompt, enter an expression representing the address or mask you want to use. The range interpretations have been previously described in Range Specifications - F7 on page 8-21. Masks were described in Masks - F8 on page 8-21.

8.8.7 Data Prompts

In address+data mode, the data qualifiers are displayed after the address qualifiers and are specified exactly the same way. In address-only mode, there are no data qualifiers.

8.8.8 Cycles Prompt

The *cycles* qualifier allows you to specify what types of bus cycles qualify to trigger the action. You select the desired cycle type or combination by using the space bar to toggle through the various selections at the *cycles* prompt. This qualifier applies to both address and data lines. The various selections and their meanings are listed below.

Table 8-8. Cycles Prompt Selections

Cycles	Meaning
ALL	Any cycle type qualifies
NONE	No cycle type qualifies (Prevents the action from ever occurring)
MR	Memory read cycles qualify
MW	Memory write cycles qualify
MR+MW	Memory reads and writes both qualify
IAQ	Only instruction fetch cycles qualify
IAQ+MR	Both instruction fetches and memory reads qualify
IAQ+MW	Both instruction fetches and memory writes qualify

You can use the F9 (Ignore) key to force the cycle type qualifier to ALL.

8.8.9 External Qualifiers Prompt

The prompt for external probe qualifiers is the last item under each action. Enter an expression that represents the 8-bit value for the external probe inputs. The external qualifiers can be masked; if enabled, the mask appears below the prompt for the qualifiers.

8.9 Local State Settings

Use the F5 (Local) key from anywhere in the BTT screen to enter the local state settings window. There are three prompts in the window, for *mode, trace mode,* and *event count.* The local state settings are for one state only; these settings may have different values in different states. The following function and control keys can be used while editing the local state settings.

Table 8-9. Local State Editing Keys

Command	Function
F6 (Global)	Move to global settings window
F10 (Help)	Help for editing local state settings
ESC (Escape)	Exit from window
UP ARROW	Move up to previous prompt in window
DOWN ARROW	Move down to next prompt in window

8.9.1 Mode of State Prompt

The first prompt in the local state window selects the mode of the state. The two choices are `ADDR ONLY` and `ADDR+DATA`. Use the space bar to cycle between these two selections.

In address-only mode, there are no data qualifiers. An event cannot be triggered on the basis of any data bus activity. In this mode, up to four actions can be allocated into the state.

In address+data mode, data qualifiers can be used to trigger actions. However, any state in address+data mode is limited to a maximum of two actions. Address+data mode is the initial default mode for all states.

If you have actions in the state, it may not be possible to switch from address only to address+data mode, because there may already be too many actions in the state. In this case, an error message is displayed. You can then go back and clear out some of the actions before you switch modes.

8.9.2 Trace Mode

The BTT has two different ways in which trace actions can be qualified: TRIX and normal. TRIX stands for "TRace Instruction acquisition eXtended." Each state can be in either normal trace mode or TRIX mode, according to a local state setting. The trace mode is the second prompt in the local state window. Use the space bar to toggle between NORMAL and TRIX mode for the state.

In normal mode, trace actions are qualified just like other actions: any bus cycle that matches ALL the qualifications for the action triggers the action. The range specifiers allow you to trace instructions within a certain address range: for example, one routine in your program. This presents a problem, however, because instructions in the specified range probably cause memory cycles outside the range and these are not traced. You get to see the instructions that executed but not all the memory cycles that occur as a result of those instructions.

TRIX mode solves this problem. In TRIX mode, the address and data qualifiers are used to qualify the IAQ (instruction fetch) cycle only. Any IAQ cycle that meets the address and data qualifications qualifies the entire instruction for tracing. Then, any cycle that occurs during the execution of the qualified instruction is traced if the cycle matches the cycle qualifier for the action.

The advantage of TRIX mode is that it not only traces instructions in a range of addresses but traces memory cycles outside that range as well. The cycles traced outside the range must be a result of instructions inside the range.

In other words, the address and data qualifiers affect only the IAQ cycle of an instruction; the cycle qualifier determines what cycles under that instruction are also traced.

8.9.3 Event Count

The last prompt in the local state window allows you to initialize the event counter for the state. The event counter counts the number of EVENT actions that occur in the state. Events are allowed to occur until the event counter reaches 0. When the counter reaches 0, the BTT either sequences to the next state, or if the current state is the end state, completes a state sequence and possibly causes a hardware breakpoint. The operation of the event counter is described more fully in the section on Events and Breakpoints on page 8-7.

The event counter must be a value from 0 to 0FFFFh (65535 decimal). The default value is 1; this means that any event that occurs is enough to sequence to the next state or cause a breakpoint. A 0 disables the breakpoint function.

8.10 Global Settings

Use the F6 (Global) key from anywhere in the BTT screen to enter the global settings window. There are five prompts in the window, **delay count**, **max trace**, **end state**, **loop count**, and **time out**. The global settings affect the operation of the entire BTT, independent of the states. Table 8-10 lists the function and control keys that can be used while editing global settings.

Table 8-10. Keys for Editing Global Settings

Command	Function
F5 (Local)	Move to local settings window
F10 (Help)	Help for editing global settings
ESC (Escape)	Exit from window
UP ARROW	Move up to previous prompt in window
DOWN ARROW	Move down to next prompt in window

8.10.1 Delay Count

The first prompt in the globals window allows you to set the delay count. The delay counter is used when the BTT reaches a hardware breakpoint condition. Before the breakpoint occurs, the BTT waits for additional trace samples to be taken, according to the number in the delay count. If the delay count is 0, which is the default value, no trace samples are taken and the breakpoint occurs immediately.

The delay count allows you to see (via the trace) what happened after, or as a result of, an event. (The delay count must be greater than zero, and a breakpoint must be qualified on the event itself.)

After the event occurs, the BTT will wait until a number of samples equal to the delay count have been taken before actually halting. The delay count is also described in Section 8.3.2, Events and Breakpoints.

If you use a non-zero delay count, you must be sure that there is a TRACE action programmed into the end state (the state the BTT will be in when it halts). Otherwise, no trace samples can be taken and the delay count never expires.

In some cases, the number of trace samples taken may exceed the specified value of the delay count because the emulator can stop only on an instruction boundary. If this happens, you will lose some of your trace samples. The delay count must be a number from 0 to 7FFh (2047 decimal).

8.10.2 Max Trace

The trace buffer on the BTT can hold a maximum of 7FFh (2047 decimal) trace samples. Normally, if more than this number of samples is collected, the buffer wraps around and the new samples overwrite the old ones. This corresponds to a max trace value of 0, which indicates that trace samples are to be collected indefinitely. Zero is the default value for this setting.

A nonzero value for the max trace setting directs the BTT to halt after it has collected that number of trace samples. When the specified number of TRACE actions occur, the BTT halts on the next instruction boundary. The status line on the screen displays the message TRACE FULL to indicate that the halt occurred because max trace has been reached.

The max trace value is specified as the second prompt in the globals window. The value must be in the range of 0–7FFh (2047 decimal).

8.10.3 End State

Of the four states in the BTT, one must be specified as the end state. The end state is the end-of-sequence state for EVENT actions. The BTT must be in the end state for a hardware breakpoint to occur. How the end state affects the operation of the BTT is described more fully in the section on Events and Breakpoints on page 8-7.

The end state defaults to the first state, state 0. With this value, the BTT will not sequence to state 1 if the event count expires. The end state is the third prompt in the global settings window. To change the end state, enter a value of 0–3 at the prompt.

8.10.4 Loop Count

The loop count is another global counter that affects hardware breakpoints. The loop count is used to repeat the state sequence a specified number of times before acually halting. The operation of the loop count is described in Section 8.3.2, Events and Breakpoints.

The default value of the loop count is 1. This means that the state sequence will occur only one time. When the event count expires in the end state, the hardware breakpoint is enabled, the delay count takes effect, and the BTT halts as soon as the delay count expires.

To change the loop count, move to the fourth prompt in the global window and enter a value in the range 0–0FFFFh (65535 decimal).

8.10.5 Time Out

The BTT has a free-running timer that starts when the BTT is reset (when the CPU starts running). This timer is sampled each time a TRACE action occurs, and the value from the timer is included in the trace sample.

The free-running timer can also be used to halt the CPU after it has run for a specified period of time. If you specify a non-zero value for the time out setting, the BTT will halt immediately when the timer reaches the specified value. If this occurs, the status line of the debugger displays `TIME OUT` as the reason for halting.

The default value for the time out is 0, which allows the CPU to run indefinitely. The time out value is the fifth prompt in the global settings window. The value can be changed by moving to the fifth prompt and entering a new constant. Specify the time constant in terms of seconds, milliseconds, microseconds and nanoseconds.

Section 9

Debugger Configuration

The debugger has a separate screen for displaying and editing various control settings that affect the overall operation of the debugger. This screen is accessed through the **C** (Config) command of the top level command menu. The configuration screen allows you to:

- Set the emulator to "continuous" run mode.
- Allow auto-update of the screen while the CPU is running.
- Set the clock source for the TMS370 CPU.
- Display the period of the current clock source.
- Display the current mode of the TMS370 device.
- Display whether or not the BTT board is installed.
- Access the memory map screen to configure the emulator memory.
- Access the color configuration utility to setup the screen colors.

This section includes the following topics:

The configuration screen is pictured in Figure 9-1. The screen consists simply of the command line, function key line, and one small window. Those configuration settings that can be edited are displayed inside the window, and the three values that cannot be edited are displayed outside the window.

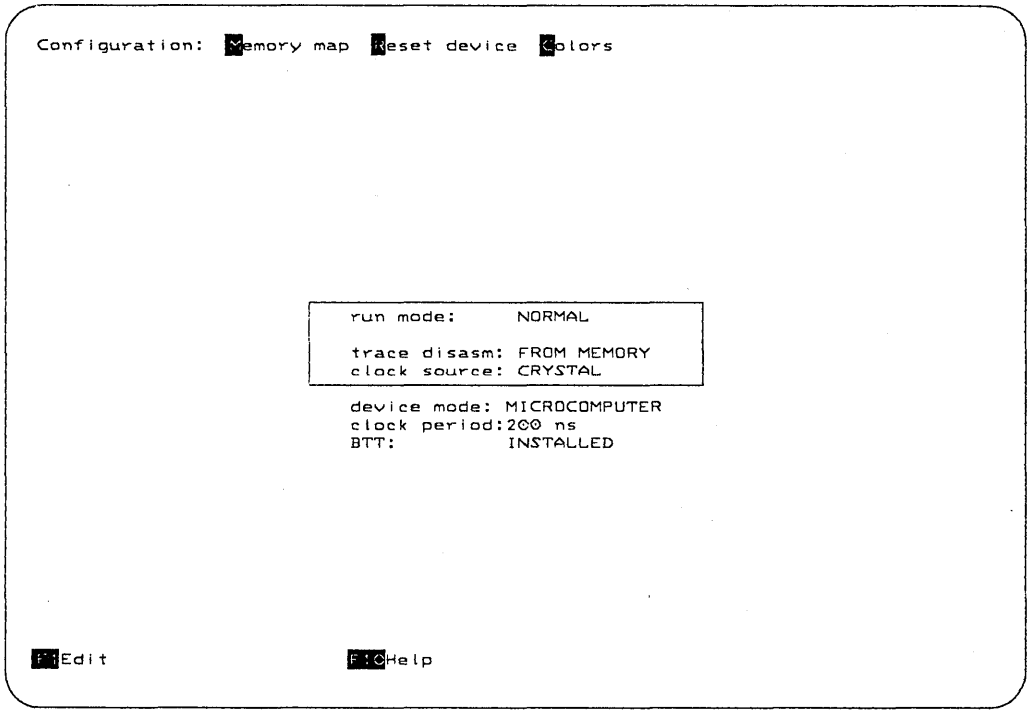


Figure 9-1. Debugger Configuration Screen

9.1 Configuration Command Menu

Table 9-1 summarizes the commands and functions available from the configuration command submenu. These functions are described in detail in the following sections.

Table 9-1. Configuration Commands

Command	Function
M (Memory Map)	Go to the memory map screen
R (Reset)	Reset the TMS370 device
C (Colors)	Go to the color setup utility
ESC (Escape)	Return to the main screen of the debugger
F1 (Edit)	Move to the window to edit the values there
F10 (Help)	Help for the config screen

9.1.1 Configure the Memory Map - The M Command

The **M** (Memory Map) command in the configuration menu exits from the configuration screen and invokes the memory map screen, which is described in Section 10, Memory Mapping.

9.1.2 Reset CPU - The R Command

The **R** (Reset) command causes the reset pin on the TMS370 device to be activated. This provides a way to reset the chip independently of any reset circuitry in the target system. The reset command also allows the debugger to update the device mode field on the screen, since this field is updated only when the debugger initiates the reset. Otherwise, the debugger cannot detect when a reset has occurred, and the mode display may be inaccurate.

9.1.3 Setup Screen Colors - The C Command

The **C** (Colors) command exits from the configuration screen and invokes the color setup utility for the debugger. This utility allows you to define the colors and display attributes used throughout the debugger. The debugger setup utility is described in Section 9.3, Debugger Color Setup.

9.1.4 Edit Configuration Settings - F1

The F1 (Edit) function key moves the cursor to the window on the screen and allows you to edit the settings there. The meaning and editing of these settings are described in Section 9.2.

9.2 Configuration Settings

There are three items on the configuration screen that are user-selectable: run mode, source of trace disassembly, and clock source.

9.2.1 Run Mode

The emulator has two different modes under which the CPU can execute code. In **normal** run mode, the emulator cannot perform any other operation while the CPU is running. The BTT and CPU run synchronously; they are started together and a halt condition on either causes both to halt. Normal run mode allows the CPU to single-step, executing one instruction at a time.

The other mode is called **continuous** run mode. In this mode, the BTT can be started and stopped independently of the CPU. The CPU can continue to run while the BTT is stopped. In this case, the emulator can perform nearly all the operations that are normally possible, such as dumping the contents of the trace buffer, re-configuring the BTT, updating the screen, and modifying the contents of registers and memory. When the debugger accesses registers and memory while the CPU is running in continuous mode, the CPU is halted just long enough to perform the operation and then restarted. Single-step operation is unavailable in continuous run mode.

The first prompt in the configuration window allows you to select the current run mode. Use the space bar to toggle between NORMAL and CONTINUOUS and make the desired selection by pressing <CR>.

9.2.2 Auto-Update

If the current run mode is *continuous*, the second prompt in the configuration window allows you to toggle the auto-update option. (In normal run mode, auto-update is not available so the prompt does not appear in the window.) Auto-update takes effect when you return to the main screen and start the CPU running. Whenever the CPU is running and the BTT is halted, the main screen is repeatedly updated with new data from the emulator. This allows you to watch your program as it runs. Updates occur appoximately 1-3 times per second, depending on the speed of your host machine and the amount of information on the screen.

Auto-update can be enabled or disabled by pressing the space bar to toggle between ON and OFF in response to the prompt. Make a selection, then press <CR>.

Auto-update significantly affects realtime operation of your program. The program still runs continuously, but the emulator constantly interrupts it to update the screen. Auto-update should not be used if realtime operation is critical.

The screen is not updated all at the same time; several instructions execute during each update cycle. For this reason, information on the screen may not appear to be consistent. For example, the same register displayed at two different places on the screen may appear as having two different values, because the running program changed its value in between updates.

While the screen is updating, you can still enter commands to the debugger. The debugger processes your commands between screen updates. The result is that response to commands appears to be sluggish while the screen is being auto-updated.

9.2.3 Trace Disassemble

This prompt allows you to decide how the trace sample will be disassembled. Use the space bar to toggle between the two choices, and press <CR> to make the selection. The choices are:

From memory: trace sample is disassembled from data read from memory.

From trace: trace sample is disassembled from data obtained from the trace buffer. Use this selection if the traced code is no longer in memory.

9.2.4 Clock Source

The final prompt in the configuration window allows you to select the clock source for the CPU. Use the space bar to toggle between the three choices, and press <CR> to make the selection. The choices are:

Crystal A crystal in the emulator is used to generate the clock.

Oscillator An oscillator circuit on the emulator is used.

Target The CPU expects the clock input from the target system.

There is no clock buffer circuitry in the target cable head, so the external clock must be a TTL compatible clock. A crystal will not work.

If you change the clock source, the debugger tries to reset the TMS370 device with the new clock source. If the emulator detects that the device is not operating properly, it rejects the new clock source and resets the chip again, restoring the old clock. When this occurs, an error message is displayed:

```
<x> not available for clock
```

where <x> is the clock source you tried to select.

9.2.5 Additional Configuration Values

Three additional items are displayed on the configuration screen, just below the window of configuration settings. The first of these indicates the device mode and can be either MICROPROCESSOR or MICROCOMPUTER. The device mode is read and updated whenever the debugger resets the TMS370 device, either by the **R** (Reset) command or by changing the clock source. The device mode is not updated at any other time; the debugger cannot detect external resets of the device. The second displayed value indicates the clock cycle period, in nanoseconds. The final item indicates whether the BTT board is installed in the emulator or not.

9.3 Debugger Color Setup

A utility program called DBSETUP, provided with the TMS370 debugger, allows configuration of the display attributes used by the debugger to display the screen. Display attributes consist of color and various effects such as highlighting, reverse video, and underlining. Using DBSETUP, the screen attributes can be customized to fit personal tastes and the capabilities of the display device in use.

DBSETUP stores attribute data in a file called `colors.dbr`. When the debugger initializes, it reads that file and uses the attributes stored there. If the file does not exist, it is created and a default set of attributes is written into it.

`Colors.dbr` can be stored anywhere on the host PC. Both the debugger and DBSETUP look for the symbol `IPCDIR` in the DOS environment table to locate the directory in which `colors.dbr` resides. If the symbol is found, its value is used for the directory; if not, the current directory is used.

For example, assume you want to keep the colors file in a directory with the pathname `c:\debug\config\colors.dbr`. Use the DOS command *SET* to define the environment symbol as follows:

```
C> set IPCDIR=c:\debug\config
```

Now, when you use the debugger or DBSETUP, the `colors.dbr` file will be looked for in the directory `c:\debug\config`. You can put the *SET* command in your `autoexec.bat` file to initialize the symbol each time the system is booted.

DBSETUP allows interactive modification of the attribute data in this file. DBSETUP can be run stand-alone from the operating system or directly from the debugger by using the **C** (Config) command at the top level command menu followed by the **C** (Colors) subcommand. When exiting DBSETUP after having run it from the debugger, the `colors.dbr` file is re-read and the screen displayed using the new attributes.

9.4 The DBSETUP Screen

The DBSETUP screen is pictured in Figure 9-2. The top line of the screen is a command menu similar to the other command menus in the debugger. The bottom line is a list of function keys; each key represents a color. To the right of the center of the screen is a miniature mock-up of the main debugger screen. To the left of that, a list of *field descriptors* is displayed. Each field descriptor corresponds to some part of the debugger screen and has a separate attribute associated with it. The field descriptors and the parts of the mock-up screen that correspond to them are displayed with their corresponding attributes. This way, you can see how the real screen will look.

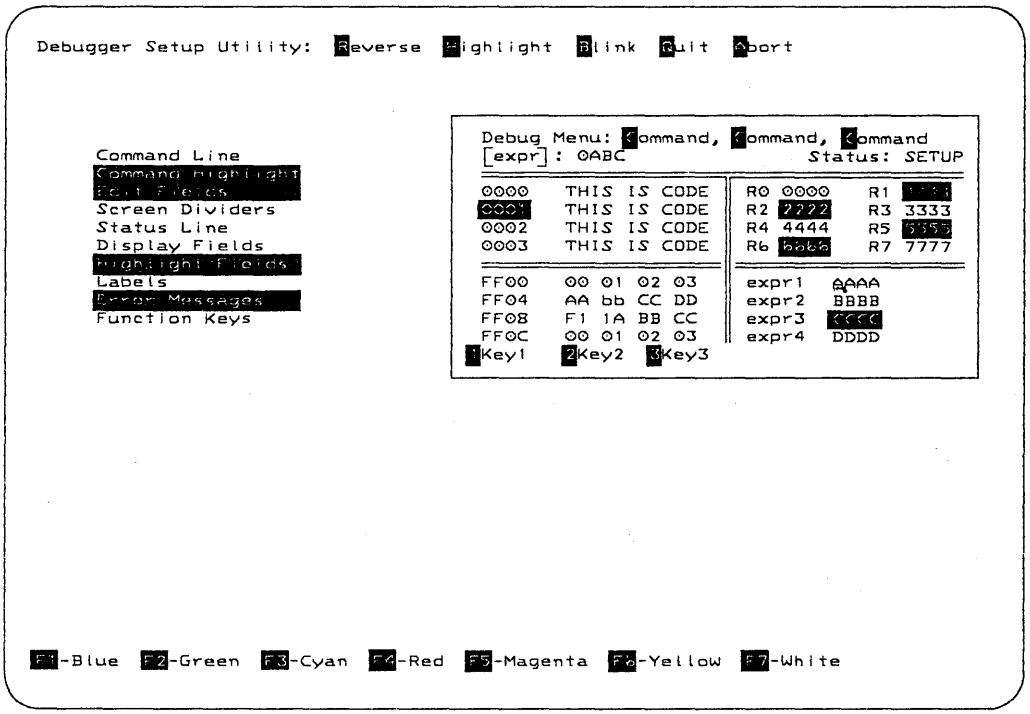


Figure 9-2. DBSETUP Screen

The various field descriptors and what they represent on the screen are described in Table 9-2 on the next page.

Table 9-2. Field Descriptors

Field	Description
Command Line	Top-line menu titles and command names.
Command Highlight	Selection letters on menus, and function keys.
Edit Fields	Current output buffer for field editor.
Screen Dividers	Double lines that delimit the windows.
Status Line	All messages and status displays on second line of screen.
Display Fields	Any displayed values. Trace samples, disassembly. BTT qualifiers and settings. Memory map and configuration settings.
Highlight Fields	Highlighted fields such as current PC, values that have changed since last update. Also, current input buffer for field editor.
Labels	Names that identify displayed values. Also, IAQ trace samples.
Error Messages	Messages displayed on bottom line of the screen.
Function Keys	Function key command names on the bottom line of the screen.

9.5 DBSETUP Operation

The cursor is positioned just to the left of the field descriptors. The cursor can be moved up and down through the descriptors using the arrow keys. At each field descriptor, you can set the color using a function key or perform various other commands as described by Table 9-3.

Table 9-3. DBSETUP Commands

Field	Description
R (Reverse)	Toggle reverse video for current field
H (Highlight)	Toggle highlight (bright) attribute for current field (IBM PC only)
B (Blink)	Toggle blink attribute for current field
Q (Quit)	Quit, saving attribute data in "colors.dbr"
A (Abort)	Quit, but do not save attribute data
ESC (Escape)	Quit, with save (same as 'Q')
F1 (Blue)	Set color of current field to blue
F2 (Green)	Set color of current field to green
F3 (Cyan)	Set color of current field to cyan
F4 (Red)	Set color of current field to red
F5 (Magenta)	Set color of current field to magenta
F6 (Yellow)	Set color of current field to yellow
F7 (White)	Set color of current field to white
UP ARROW	Move up to the previous field descriptor
DOWN ARROW	Move down to the next field descriptor

Note:

On an IBM PC with a monochrome display board, all the colors are displayed as white, except for blue, which appears as underlined white. Also, the **R** (Reverse) attribute is visible only if the selected color for the field is white.

Use the commands and functions above to move between the field descriptors and select attributes for them. When you change an attribute, the field descriptor and corresponding parts of the mock-up screen are immediately displayed with the new attribute.

When finished, type **Q** (Quit). The new attributes are stored in the file `colors.dbr`. If you have made changes but do not want to save them, exit with **A** (Abort).

If you called DBSETUP from the debugger, you return to the debugger. The debugger reads the new colors file and updates the screen to reflect any changes made.

Section 10

Memory Mapping

The memory mapping capability of the TMS370 debugger allows you to specify, for various ranges of the address space, exactly how memory accesses are satisfied by the CPU. The TMS370 device provides significant flexibility in the memory map, with its various configurations of internal memory and external addressing capability. In addition, the emulator has 64K bytes of high-speed emulation RAM, which can be used to functionally emulate target system memory. The debugger memory mapping system allows you to take advantage of the hardware's flexibility.

Specifying the memory map is accomplished by dividing the 64K address space into ranges. Each range starts and ends on a 16-byte boundary and has three attributes that determine how addresses in the range are accessed. The three attributes are type, mapping, and protection.

The type attribute for a memory range specifies what class of memory the range falls into, such as register file, program ROM, EEPROM, etc. The type for a given range is determined by the architecture of the particular TMS370 device being emulated. The mapping attribute specifies, for a given type, whether the memory is to be accessed internally on the chip, from external target memory, or from emulator RAM. Finally, the protection attribute allows you to prevent the CPU from reading and/or writing to the given address range.

Any address that is not covered by a specified range in the debugger memory map is considered to be *unconfigured*. All unconfigured memory is protected against both read and write operations, so that if the CPU tries to access an unconfigured location, an access violation occurs and the CPU halts.

Contents of this section include:

10.1 Memory Map Screen

The memory map screen of the debugger is pictured in Figure 10-1. The screen is divided into three areas. The first area, in the upper left part of the screen, contains specifications of the TMS370 device being emulated. These specifications define the type attribute for all addresses in the memory map. The second area, in the lower left, allows you to specify address ranges and define the attributes for them. Finally, the right half of the screen is dedicated to a "blow-up" of the map for the peripheral file. This allows you to specify mapping and protection attributes for individual frames of the peripheral file without having to explicitly specify frames as separate ranges.

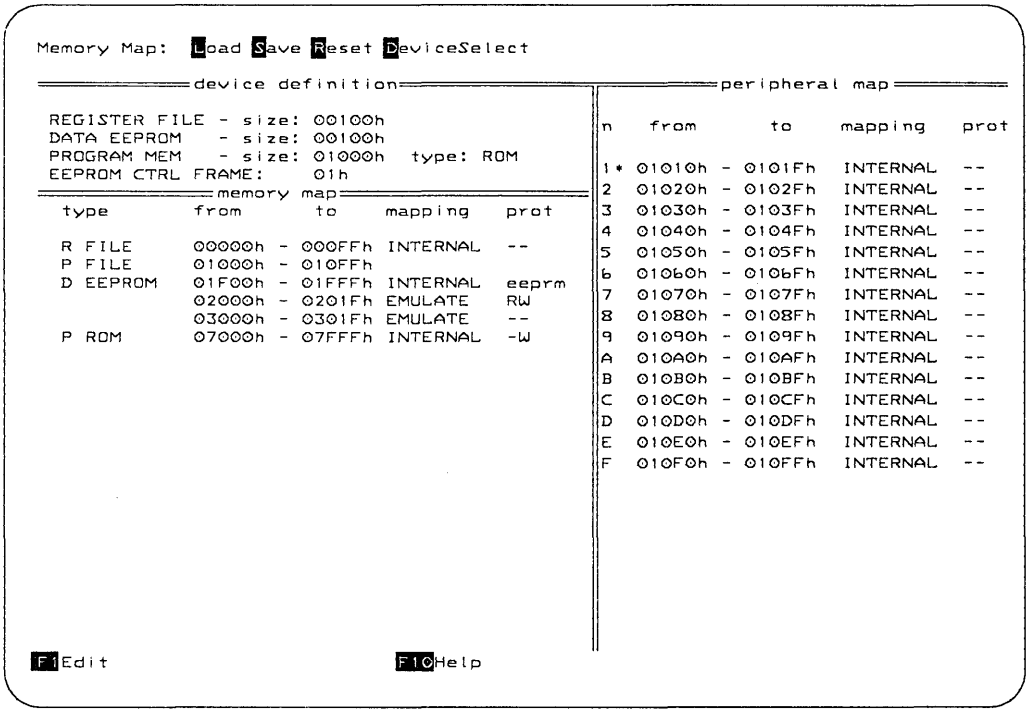
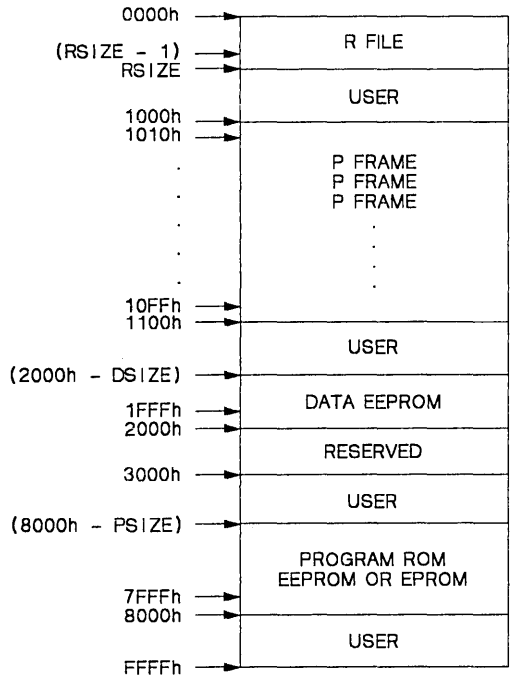


Figure 10-1. Memory Map Screen

10.2 Memory Types

Any given range in the memory map is one of eight types: REGISTER FILE, PERIPHERAL FRAME, PROGRAM ROM, PROGRAM EEPROM, PROGRAM EPROM, DATA EEPROM, EEPROM CONTROL FRAME, and USER. In the memory map window, the memory type for a given range is displayed with the range. Each type is described below. Also, the options for mapping and protecting each type of memory is given.

Figure 10-2 shows a map of the address space and how memory types are assigned to addresses according to the device definition. The device definition is described in Section 10.4, Device Definition Window.



WHERE: RSIZE = REGISTER FILE SIZE
DSIZE = DATA EEPROM SIZE
PSIZE = PROGRAM MEMORY SIZE

Figure 10-2. Device Specifications for Various Memory Maps

10.2.1 Register File Memory

Description: Locations are part of the TMS370 on-chip register file.

Mapping Attributes: The mapping attribute for the register file is always INTERNAL, meaning that accesses to these locations always occur on the device itself; no external referencing is possible.

Protection Attributes: Any combination of read and write protection can be applied to register file addresses. If the CPU accesses a protected register, an access violation trap occurs and the CPU halts.

> **Note:**
>
> A write protection violation causes the trap but does not prevent the data from being written to the register. The default mapping for the register file is no protection.

10.2.2 Peripheral Frame Memory

Description: This type is used for 16-byte ranges in the peripheral register file. These ranges appear in the right half of the screen, separate from the main memory map where they are grouped together as `P FILE`.

Mapping Attributes: By default, peripheral frames are mapped as INTERNAL, which means that accesses are to the actual peripheral registers on the TMS370 device. However, an expansion board can be added to the emulator which contains hardware for additional peripheral frames or replacement hardware for existing on-chip frames. A frame on the expansion board can be accessed by setting the mapping attribute for the frame to EXTERNAL.

Protection Attributes: Any combination of read and write protection can be applied to peripheral file addresses. If the CPU accesses a protected register, an access violation trap occurs and the CPU halts.

10.2.3 Program ROM Memory

Description: This memory type is used for addresses that fall in the range of the on-chip mask ROM. If the program memory space of the current device is EEPROM, there is no memory of this type in the map. Program ROM is emulated using the high-speed RAM in the emulator; the TMS370 device used in the emulator has no actual on-chip ROM of its own.

Mapping Attributes: Program ROM is always mapped as INTERNAL, which means that whenever internal memory is enabled on the device, memory accesses in this range are satisfied by the emulator (just as if the memory were actually on the chip). If internal memory is disabled, such as when the device is in microprocessor mode, accesses in this range are to external memory on the target system.

Protection Attributes: Program ROM is, by default, write protected. Write cycles to addresses in a range of this type cause an access violation trap to occur. If internal memory is enabled, which means that data would be written to emulator RAM, the data is not written (in other words, emulator RAM that

is write protected cannot be written to). Writes to external target memory cause a trap, but the write operation cannot be prevented.

Write protection is the default for Program ROM, although the protection attribute can be changed. You can remove the write protection and have no protection, or you can add read protection so that all or part of the program ROM space is completely protected.

10.2.4 Program EPROM Memory

Description: Program EPROM memory is similar to program EEPROM memory with the following differences:

1) Only write to zero is supported (W1W0=0).

2) No block erase feature is supported.

Mapping Attributes: See Program EEPROM above.

Protection Attributes: You cannot change the protection attributes of EPROM. On the memory map screen, EPROM is displayed as **eprm** to indicate the special treatment of this memory range. Write cycles to EPROM memory do not cause access violation traps which halt the CPU.

10.2.5 Program EEPROM Memory

Description: The Program EEPROM type is used for addresses that fall in the range of the on-chip program EEPROM. EEPROM is emulated using the high-speed RAM in the emulator; the TMS370 device used in the emulator has no actual on-chip EEPROM of its own.

The emulator simulates the operation of the EEPROM by trapping internally whenever the EEPROM or its control registers are accessed, deciding how the real EEPROM would behave, and then simulating that behavior. Thus, EEPROM emulation is not accurate, and there are some side effects of this scheme that could cause problems in certain target environments.

First, since EEPROM is simulated by software in the emulator, EEPROM operations are not realtime. The software emulates an EEPROM write by halting the processor for several milliseconds. This delay occurs during any memory access to the EEPROM Control Frame or any write to EEPROM memory. If the delay is not wanted, change the EEPROM size to 0 and redefine a new block of memory in the memory map window.

Second, no mapping or protection options are allowed for EEPROM memory types. Finally, EEPROM simulation is always in effect, even if internal memory in the device is disabled. In other words, if your application is switching between internal and external memory, the program memory should not be defined as EEPROM or accesses to external target memory will never occur. The best way around this limitation is to define the program memory space as ROM or USER, so that mode changes are more accurately emulated.

Mapping Attributes: EEPROM memory types are always mapped as INTERNAL, which means that memory accesses in this range are satisfied by the emulator. Note that this is true even if internal memory is disabled (see the discussion above).

Protection Attributes: You cannot change the protection attributes of EEPROM. On the memory map screen, EEPROM is displayed as **eeprm** to indicate the special treatment of this memory range. Write cycles to EEPROM memory do not cause access violation traps which halt the CPU.

10.2.6 Data EEPROM Memory

Description: The Data EEPROM type is used for addresses that fall in the range of the on-chip data EEPROM. Some devices have no memory of this type in the memory map. Data EEPROM is emulated using the high-speed RAM in the emulator in exactly the same way as Program EEPROM. See the description above for details on how EEPROM is treated in the emulator and the restrictions on its use.

Mapping Attributes: See Program EEPROM above.

Protection Attributes: See Program EEPROM above.

10.2.7 EEPROM/EPROM Control Frame Memory

Description: This is a special memory type that is applied to the peripheral register frame in which the control registers for EEPROM/EPROM are located. If no EEPROM/EPROM is present in the memory map, there will be no EEPROM/EPROM control frame. When EEPROM/EPROM is present in the map, accesses to the control registers are trapped and simulated in the same way as accesses to the EEPROM/EPROM itself.

Since mapping is on 16-byte boundaries, **ANY** write access to the frame in which the control registers are located causes this trap to occur. This trap causes the processor to halt for several milliseconds. On the actual TMS370 device, the control register is frame 1 (addresses 1010h - 101Fh). Frame 1 contains many other control registers and it is desirable to avoid taking a trap whenever any of the other registers are accessed.

To avoid this problem, you can map the EEPROM/EPROM control registers into a different frame of the peripheral file and thereby avoid causing a delay whenever frame 1 is accessed. Typically, the control registers are mapped into an unused frame. The offset for each control register in the frame remains the same. For example, the registers on the actual device are at addresses 101Ah, 101Ch, and 101Eh. If you change the EEPROM/EPROM control frame to frame 8, the emulator acts as if the registers are at addresses 108Ah, 108Ch, and 108Eh. You could modify your source code to operate on the new addresses. If you want the several-millisecond delay, use ".equ" directives in the assembly source to define the EEPROM/EPROM control registers.

The EEPROM/EPROM control frame is specified with the device parameters. The peripheral frame that is defined as the EEPROM/EPROM control frame is identified on the screen with an asterisk ('*') next to the frame number.

10.2.8 User Memory

Description: Any memory range that is not explicitly defined by the device characteristics to be one of the other types is a USER memory range. Thus, all memory addresses are either USER or one of the types described above. User ranges show up in the memory map with no "type" field. USER memory allows you to configure other regions of the address space as being accessible to the CPU, in addition to those implicitly configured by the device type.

Mapping Attributes: USER memory can be mapped as EMULATE or TARGET. When an address range is mapped as EMULATE, the emulator uses the on-board emulation RAM for accesses to those addresses in the range. When mapped as target, the emulator expects the target system to satisfy memory cycles to addresses in the range. You can use EMULATE as a substitute for target memory when the target memory circuitry is missing or incomplete.

Protection Attributes: Any combination of read and write protection is available for use with USER ranges. By default, all USER memory is fully protected against both read and write access, thus causing an access violation trap to occur when the CPU attempts a cycle. Such default ranges do not appear in the memory map display unless you explicity specify them for inclusion in the window. Then, you can remove all protection, specify read protection only, or specify write protection only.

> **Note:**
>
> If a write protection violation occurs with memory that is mapped to TARGET, the emulator cannot prevent the data from being written out (the trap occurs, but the write is not prevented). The write cycle is prevented only if the address range is mapped to EMULATE.

10.3 Top Level Memory Map Commands

You invoke the memory map screen from the top level command menu by typing **CM** (Config,Memory). The current memory map configuration is displayed in the three areas of the screen, an example of which is shown in Figure 10-3. The memory map screen has its own menu of commands, which are used for top level memory map functions. You can also move to each of the three windows to perform editing operations.

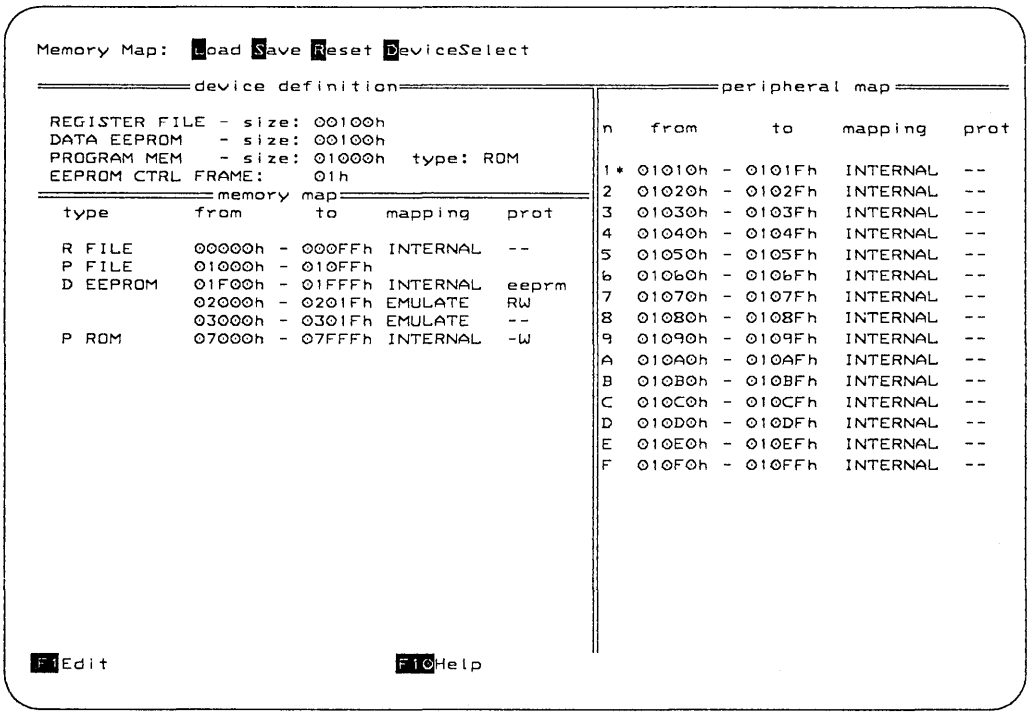


Figure 10-3. Memory Map Configuration Screen

Table 10-1 describes the commands and functions for the memory map command menu.

Table 10-1. Memory Map Commands

Command	Function
L (Load)	Load the memory map from a file
S (Save)	Save the current configuration in a file
R (Reset)	Reset all ranges to default
D (Device Select)	Define map for "standard" TMS370 device
ESC (Escape)	Return to the main screen
F1 (Edit)	Move to the device definition window
F2 (Last)	Move to the peripheral file window
F10 (Help)	Help for the memory map screen

10.3.1 Load Map from File - The L Command

The **L** (Load) command in the memory map command menu allows you to load a file containing memory map information and set up the memory map according to the information in the file. Memory map files contain the settings from all three of the windows: device definition, additional attributes/ranges, and peripheral file mapping. Use the **S** (Save) command described below to create a saved memory map file. Note that loading a new memory map destroys the current map.

When you type **L**, the debugger prompts you for a filename. You can omit the extension on the filename if the default extension is `.mmp` (for Memory Map). If the file entered cannot be found or is found to be of the wrong type (memory map files have a special identification code), an error message is issued and the command aborts. Otherwise, the debugger reads the file, sets up the memory map in the emulator accordingly, and displays the newly loaded map on the screen.

Memory map files can also be loaded using the **L** (Load) command at the top level debugger menu, provided a `.mmp` extension is added. See Section 5.9, Loading a File - The L Command.

10.3.2 Save Map in File - The S Command

The **S** (Save) command saves the current memory map specification in a disk file for later retrieval using the **L** (Load) command. This allows the use of different map specifications for different debugging applications, without having to re-enter the map each time the debugger is invoked.

When the **S** command is entered, the debugger prompts for a filename. Again, the default extension is `.mmp` and should be used so that map files can be loaded from the main screen also. The debugger creates the file, deleting any existing file with the same name, and stores a representation of the current memory map in the file.

Memory map files contain the settings from all three of the windows: device definition, additional attributes/ranges, and peripheral file mapping. The file is stored with a special key to indicate that it is a memory map file and is in machine readable format only.

10.3.3 Reset Map - The R Command

The **R** (Reset) command simply restores the memory map to the basic configuration as defined by the current device definition. All ranges in the map are set to have the default attributes for their memory type. All USER ranges are removed (since the default attributes for USER memory makes it unconfigured).

The effect of this command is to "undo" anything you have done to the memory map in the map or peripheral file windows. The Reset command has no effect on the device definition settings.

10.3.4 Select a Standard Device Type - The D Command

The **D** (Device Select) command is a shortcut for specifying a set of values in the device definition window. The debugger has a table of standard device types corresponding to the TMS370 devices that are currently available. For each device, the table contains the complete set of device definition values. The device table is shown as Table 10-2.

Table 10-2. Device Definitions

Device	R File Size	D EEPROM Size	Prog Mem Size	Prog Mem Type	EEPROM Control
TMS370C010	80h	100h	1000h	ROM	1
TMS370C810	80h	100h	1000h	EEPROM	1
TMS370C050	100h	100h	1000h	ROM	1
TMS370C850	100h	100h	1000h	EEPROM	1

On initial power-up, the debugger sets up the memory map according to the definition of the TMS370C050 device.

When you type **D**, the debugger displays the name of the most recently selected device. Use the space bar to cycle through the devices in the table. Press <CR> to select one of the devices. The debugger fills the values from the selected device into the device specification window, sends the configuration to the emulator, and updates the screen to reflect the new memory map.

Selecting a device with the **D** command is exactly equivalent to moving to the device specification window and manually entering the values from a row of Table 10-2.

10.4 Device Definition Window

The values represented in this window define the memory architecture characteristics of the device being emulated. Specifically, the device specifications determine the memory type attributes for the entire address space. This is the only way that type attributes can be set or changed.

Any changes you make to the device definition do not take effect until you leave the window. Then, the changes are sent to the emulator, and the screen is updated to reflect the new configuration of the memory map. When you change the device definition, all old device definitions are removed from the map.

Any USER memory ranges are kept but may be fragmented or deleted if they overlap ranges that are imposed by the new device definition. For example, assume you have a USER range from address 6000h to 7FFFh and no program memory. Then, you modify the device definition to put 4K of Program ROM from address 7000h to 7FFFh. This causes the USER range to be fragmented, resulting in USER memory from 6000h to 6FFFh and Program ROM from 7000h to 7FFFh. This can be summarized as follows: when a new device definition is merged into the memory map, the memory ranges defined by the new device have priority over all other existing ranges in the map.

Refer to Table 10-2 to see how the device definition parameters are used to assign type attributes to the address space. Refer to Section 10.2, Memory Types, for a description of the memory type attributes and to Section 10.5.1 on page 10-15 for examples.

You edit the settings by moving to the appropriate prompt and entering a new value. The debugger produces error messages when values are out of range or otherwise invalid. Table 10-3 outlines the editing keys used to move around in the window.

Table 10-3. Device Definition Window Editing Keys

Command	Function
ESC (Escape)	Return to the main screen
F1 (Next)	Move to the memory map window
F2 (Prev)	Move to the peripheral file window
F10 (Help)	Help for the device definition window
L ARROW	Left or up to previous prompt
UP ARROW	Move up to the previous prompt
R ARROW	Right or down to next prompt
DN ARROW	Move down to the next prompt

A description of each device definition follows.

10.4.1 Register File Size

The first prompt in the device specification window defines the size of the register file. The register file can be from 128(80h) to 256(100h) bytes long, in increments of 16(10h) bytes. The register file always starts at address 0h. Any memory from 0000h through the size of the register file is assigned type REGISTER FILE.

10.4.2 Data EEPROM Size

The second prompt defines the size of the data EEPROM area on the device. The value can range from 0 to 1000h, although the largest data EEPROM area currently available on an actual device is 100h bytes. The size must be a multiple of 16 (10h) bytes. You can specify a size of 0, in which case no memory in the map is of type DATA EEPROM.

The data EEPROM area in the map grows downward from a high boundary at address 2000h. Thus, the starting address can be found by subtracting the size from 2000h.

10.4.3 Program Memory Size

The third prompt in the device definition window determines how much on-chip program memory the device has. This memory can be either mask ROM or EEPROM, depending on the value of the program memory type (next prompt). Program memory size can range from 0 to 4000h. You can specify a size of 0, in which case no memory in the map is configured for program memory.

The program memory area in the TMS370 map grows downward from a high boundary at address 8000h. The starting address of program memory is found by subtracting the size from 8000h. For devices that have 1000h bytes of program memory, the starting address is 7000h.

10.4.4 Program Memory Type

The prompt for program memory type is located next to the program memory size. If there is no program memory (size = 0), the type prompt does not appear. There are two choices for program memory type: ROM or EEPROM. See Section 10.2, Memory Types, for descriptions. Use the space bar to toggle between the two choices and press <CR> to make a selection.

10.4.5 EEPROM Control Frame

The final prompt in the device definition window is for the EEPROM control frame. If there is no EEPROM specified (data EEPROM size is 0 and program memory is ROM), this prompt does not appear. Valid values for this prompt range from 1 to 15 (0Fh), corresponding to the 16-byte frames of the peripheral file. The starting address for the frame is given by:

start address = 1000h + (n * 10h)

where n is the frame number.

Any access to any register in the EEPROM control frame will produce a several-millisecond delay in an attempt to emulate EEPROM. This delay will also affect the system configuration and interrupt registers. If you want to avoid the delays caused by this EEPROM emulation scheme, move the EEPROM control register to an unused frame; i.e., frame 08.

10.5 Memory Map Window

The memory map window, located in the lower left part of the screen, displays a representation of the memory map for the entire address space. All configured memory is represented in this window. The map is displayed as a list of address ranges, displayed as follows: the type attribute, the addresses in the range, the mapping attribute, and the protection attributes. Ranges that have no type attribute are USER ranges.

The entire peripheral file is represented in the map as a range with a dummy type of P FILE; this range is simply a place holder for the peripheral file addresses. The P FILE range has no attributes in the memory map window and cannot be edited, since attributes for the peripheral file can be specified on a frame-by-frame basis in the separate peripheral file map window.

In the memory map window, you can define new ranges, remove ranges, or move up and down through the list and modify attributes, under the control of a command menu. The current range is highlighted. Table 10-4 summarizes the commands and functions used to inspect the memory map.

The options available for editing attributes for a given range depend on the memory type of a range. For example, only protection attributes can be specified for an R FILE range. For an EEPROM range, no editing of attributes is allowed at all. As you move up and down through the list of memory ranges, the command menu changes to contain only the options that are available for the current range.

Table 10-4. Inspect Memory Commands

Command	Function
A (Add)	Define a new memory range
D (Delete)	Restore default attributes to a range
R (Read Protect)	Add read protection to a range
W (Write Protect)	Add write protection to a range
N (No Protection)	Remove all protection from a range
E (Emulate)	Set mapping attribute to EMULATE
T (Target)	Set mapping attribute to TARGET
ESC (Escape)	Return to the memory map top level
F1 (Next)	Move to the peripheral file window
F2 (Prev)	Move to the device definition window
F10 (Help)	Help for the memory map window
UP ARROW	Move up to the previous range
DN ARROW	Move down to the next range

10.5.1 Adding a New Range - The A Command

The **A** (Add) command allows you to add new ranges to the memory map. When a new range is added, it is first assigned a type attribute according to where in the address space the range falls. For example, assume there is a range of Program ROM from 7000h to 7FFFh. If a new range is added within this range, its type is also Program ROM. If the new range does not fall within any existing configured range, it becomes a USER range.

Ranges in the memory map cannot straddle type boundaries. If a new range is added that overlaps two or more ranges of different types, the new range is fragmented into multiple ranges having different types.

The mapping and protection attributes for a newly added range are also set to be the same as whatever the new range overlaps. Thus, adding a new range has no direct effect on the memory map itself. Adding a range simply enters the range as a distinct part of the memory map, so that its mapping and protection attributes can subsequently be modified.

Assume the current memory map is as follows:

```
type       from      to       mapping   prot

R FILE     00000h - 000FFh   INTERNAL  --
P FILE     01000h - 010FFh
D EEPROM   01F00h - 01FFFh   INTERNAL  -W
P ROM      07000h - 07FFFh   INTERNAL  -W
```

Example 1. Add a range from 10h to 0FFh. The range falls completely within the R FILE, so it gets type R FILE. The existing R FILE is fragmented to accommodate the new range:

```
type       from      to       mapping   prot

R FILE     00000h - 0000Fh   INTERNAL  --
R FILE     00010h - 000FFh   INTERNAL  --   (new range)
P FILE     01000h - 010FFh
D EEPROM   01F00h - 01FFFh   INTERNAL  -W
P ROM      07000h - 07FFFh   INTERNAL  -W
```

Example 2. Add a range from 6000h to 70FFh. The new range overlaps unconfigured memory from 6000h to 6FFFh and P ROM from 7000h to 70FFh. The new range is broken into two ranges. The first is a USER range (because it overlaps unconfigured memory) and the second is P ROM.

```
type       from      to       mapping   prot

R FILE     00000h - 000FFh   INTERNAL  --
P FILE     01000h - 010FFh
D EEPROM   01F00h - 01FFFh   INTERNAL  -W
           06000h - 06FFFh   TARGET    RW (* new range 1 *)
P ROM      07000h - 070FFh   INTERNAL  -W (* new range 2 *)
P ROM      07100h - 07FFFh   INTERNAL  -W
```

Use the **A** (Add) command to add ranges in the manner described above. The debugger prompts for the starting and ending addresses - enter expressions for each of these prompts. The start address must be on a 16-byte boundary, and the end address must be one less than a 16-byte boundary. If either of these conditions is not met, the debugger rounds the addresses up to the next

satisfactory boundary. The debugger also ensures that the start address is less than the end address.

If all is in order, the debugger adds the range (or ranges, if the new range gets fragmented) according to the rules described above. The memory map is re-displayed to reflect the addition of the new range. The cursor is positioned at the new range so that you can now define mapping and protection attributes for it.

10.5.2 Reset a Range to Default/Delete - The D Command

Use the **D** (Delete) command to reset the mapping and protection attributes for the current memory range back to their default values according to the memory type. This has the effect of undoing any changes you have made to the mapping or protection of the range. In this way, you can remove non-default ranges (such as USER ranges) from the memory map.

For example, assume the memory map is as follows:

```
type       from      to       mapping   prot

R FILE     00000h - 000FFh  INTERNAL  --
P FILE     01000h - 010FFh
D EEPROM   01F00h - 01FFFh  INTERNAL  -W
           06000h - 06FFFh  EMULATE   --  <=== current
P ROM      07000h - 070FFh  INTERNAL  RW
P ROM      07100h - 07FFFh  INTERNAL  -W
```

The current range is USER memory that has been mapped into emulator memory and is completely unprotected. If you type 'D' while positioned at this range, the mapping is changed back to target and all protection is restored (this is the default for USER memory). The USER map is removed, resulting in:

```
type       from      to       mapping   prot

R FILE     00000h - 000FFh  INTERNAL  --
P FILE     01000h - 010FFh
D EEPROM   01F00h - 01FFFh  INTERNAL  -W
P ROM      07000h - 070FFh  INTERNAL  RW  <=== current
P ROM      07100h - 07FFFh  INTERNAL  -W
```

Now, the current range is ROM that is completely protected. If you type **D** again, the protection is reset to the default (write-protect only). The deleted range is then merged into the rest of P ROM, as follows:

```
type       from      to       mapping   prot

R FILE     00000h - 000FFh  INTERNAL  --
P FILE     01000h - 010FFh
D EEPROM   01F00h - 01FFFh  INTERNAL  -W
P ROM      07000h - 07FFFh  INTERNAL  -W
```

10.5.3 Modify Protection - The R, W, and N Commands

The **R** (ReadProtect) command protects the current range so that any read operation from the CPU in the specified address range causes an access violation. Similarly, the **W** (WriteProtect) command protects the current range from write operations. If the CPU executes a write cycle to memory that is write protected, an access violation occurs.

In some cases, the emulator cannot actually prevent data from being written to memory when a write violation occurs (the violation is detected but not prevented). These cases depend on the memory type and where it is mapped. In general, only write operations to internal emulator RAM can be prevented; others are only detected.

Internal RAM is used for the following memory types: Program ROM, all EEPROM, and USER memory that is mapped to EMULATE. Writes to memory of type R FILE, P FILE, or any memory that is mapped to TARGET can never be prevented.

The **N** (NoProtect) command simply removes all protection from the current memory range.

10.5.4 Modify Mapping Attributes - The E and T Commands

Use the **E** (Emulate) command to map the current range into emulator memory. By doing this, you can "emulate" the memory of the target system. The emulator satisfies references to external memory that fall in the current range using its on-board high-speed emulation RAM, rather than going to the target system.

Use the **T** (Target) command to map the range back to target memory. The emulator expects the target system to satisfy all external memory accesses that fall within the range.

10.6 Peripheral File Window

The final window on the memory map screen is located on the right half of the screen and contains a blown-up representaion of the map for the peripheral file. Each of the 15 peripheral file frames is represented by one line in the window (frame 0 is omitted since its map is not available for modification). Each 16-byte frame is treated as a separate range and behaves in a way similar to the ranges in the main memory map window.

The frame at which the cursor is positioned is the *current frame*. Like the memory map window, you can change the mapping and protection attributes of the current frame or move between the frames using the arrow keys. Ranges cannot be added or deleted from the peripheral file map.

The EEPROM control frame is displayed with an asterisk ("*") next to the frame number. You cannot change the mapping or protection of the EEPROM control frame.

The command menu for inspecting the peripheral frames is similar to the one for the main memory map. Table 10-5 summarizes the available commands and functions.

Table 10-5. Peripheral File Window Commands

Command	Function
I (Internal)	Set mapping attribute to INTERNAL
E (Expansion)	Set mapping attribute to EXPANSION
R (Read Protect)	Add read protection to a frame
W (Write Protect)	Add write protection to a frame
N (No Protection)	Remove all protection from a frame
ESC (Escape)	Return to the memory map top level
F1 (Next)	Move to the device definition window
F2 (Prev)	Move to the memory map window
F10 (Help)	Help for the peripheral file map window
UP ARROW	Move up to the previous frame
DN ARROW	Move down to the next frame

10.6.1 Modify Mapping Attributes - The E and I Commands

The emulator supports an add-on peripheral expansion board, which can be used to realize additional custom peripheral modules. By mapping peripheral frames to the expansion board, the emulator can access I/O addresses on the board as if the custom module was a normal on-chip peripheral frame.

Use the **E** (Expansion) command to map the current frame onto the expansion board. The emulator then expects the board to satisfy all peripheral file accesses in the frame.

Use the **I** (Internal) command to map the frame back to normal, so that accesses to the frame occur from within the TMS370 device.

10.6.2 Modify Protection - The R, W, and N Commands

The **R** (ReadProtect) command protects the current frame so that a read operation from the CPU to any register in the frame causes an access violation. Similarly, the **W** (WriteProtect) command protects the current frame from write operations. If the CPU executes a write cycle to memory that is write protected, an access violation occurs. The emulator cannot prevent the data from being written to the register. It simply detects the violation and halts the CPU.

The **N** (NoProtect) command simply removes all protection from the current peripheral frame.

The protection commands can be used to *unconfigure* peripheral frames that are non-existent. Simply use **R** and **W** to fully protect any frame that is not associated with an actual peripheral module. This way, you can detect erroneous accesses to nonexistent registers.

Section 11

The XDS System

This section contains installation and operation information for an XDS/22 System containing an XDS/370 Emulator for program development. Emulator requirements and a summary of startup procedures are presented. Operation of the emulator is carried out primarily through use of the software described in the first part of this manual.

The following topics are included in this section:

Warning:

Since many of the procedures described in this section require a working knowledge of computer and electronic equipment, Texas Instruments recommends that all installations and procedures be made by a trained technician.

11.1 Chassis Description

This section describes the XDS/22 unit chassis.

An XDS/22 unit cabinet mounts an operator panel containing switches and indicator lights; a rear panel containing port connectors, line-cord connector and fuse holder, and an exhaust fan; a top cover; and a front card-cage cover.

The cabinet contains a power supply, seven-slot card cage with backplane connector, and chassis cable-mounting bracket.

This section includes the following topics:

11.1.1 Operator Panel

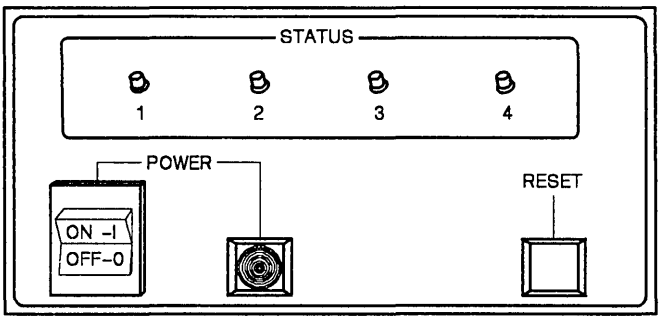


Figure 11-1. XDS/22 Front Panel Controls and Indicators

The controls and indicators on the XDS/22 Operator Panel (Figure 11-1) function as follows:

- ON/OFF POWER Switch: Applies line voltage to the unit.
- POWER Indicator: Lights when power is applied to the unit.
- RESET Switch: When pressed, sends a hardware reset signal to the I/O devices, TMS370, and multiplexer devices on the TMS370 Emulator card. There is a slight delay as voltage is sampled to ensure that it is within an acceptable range.
- STATUS Indicators: Display emulator status.

  1) Status Indicator #1, IDLE. This light comes on when the TMS370 is in the idle mode, which indicates that the processor is executing

an IDLE instruction. The idle indicator resets upon an interrupt acknowledge or when the emulator enters the control mode.

2) Status Indicator #2, ILLEGAL OPCODE DETECT. This indicator lights when the emulator detects that the TMS370 has fetched an invalid instruction.

3) Status Indicator #3, not used by the TMS370 XDS.

4) Status Indicator #4, RUNNING. This indicator lights when the emulator executes user code.

11.1.2 Rear Panel

The rear panel (Figure 11-2) houses the four female DB-25 port connectors, line cord/fuse connector, and exhaust fan.

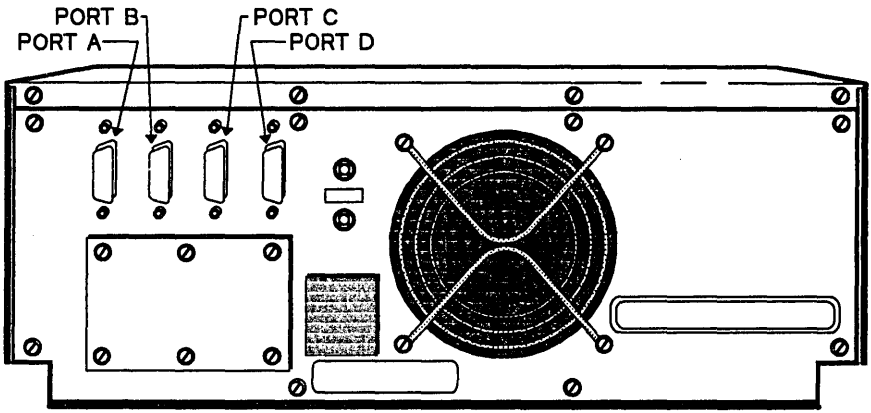


Figure 11-2. Rear of XDS/22 Unit

The line cord/fuse holder accepts 3AG or 5mm x 20mm fuses. The mechanical design requires that you pull the line cord before replacing the fuse. For International applications (220 or 240 V, 50 Hz), the holder also includes a special line filter (TI Part No. 22221618-0001) to meet VDE[4] requirements.

The exhaust fan pulls cooling air through the ventilation slots and air filters in the top cover.

The pinout for the DB-25 connector labeled Port A corresponds to Table 2-3 on page 2-6, and is described in Section 2.4, Emulator/Debugger Cabling.

---

4 VDE (Vereinigung Deutscher Electrizitaetswerke) sets electrical standards for equipment sold in the International market, just as Underwriters Laboratory (UL) sets standards for the United States.

11.1.3 Card Cage and Cover

The various emulator cards plug into the card-cage backplane connector in the order shown in Table 11-1.

Table 11-1. XDS/22 Card-Cage Slot Assignments

SLOT	PART DESCRIPTION
7	Not used
6	Not used
5	Not used
4	Memory Expansion/Communications card
3	Reserved for emulator card expansion
2	TMS370 Emulator card
1	Breakpoint/Trace/Timing card

The XDS/22 chassis with a full complement of options is shown in Figure 11-3.

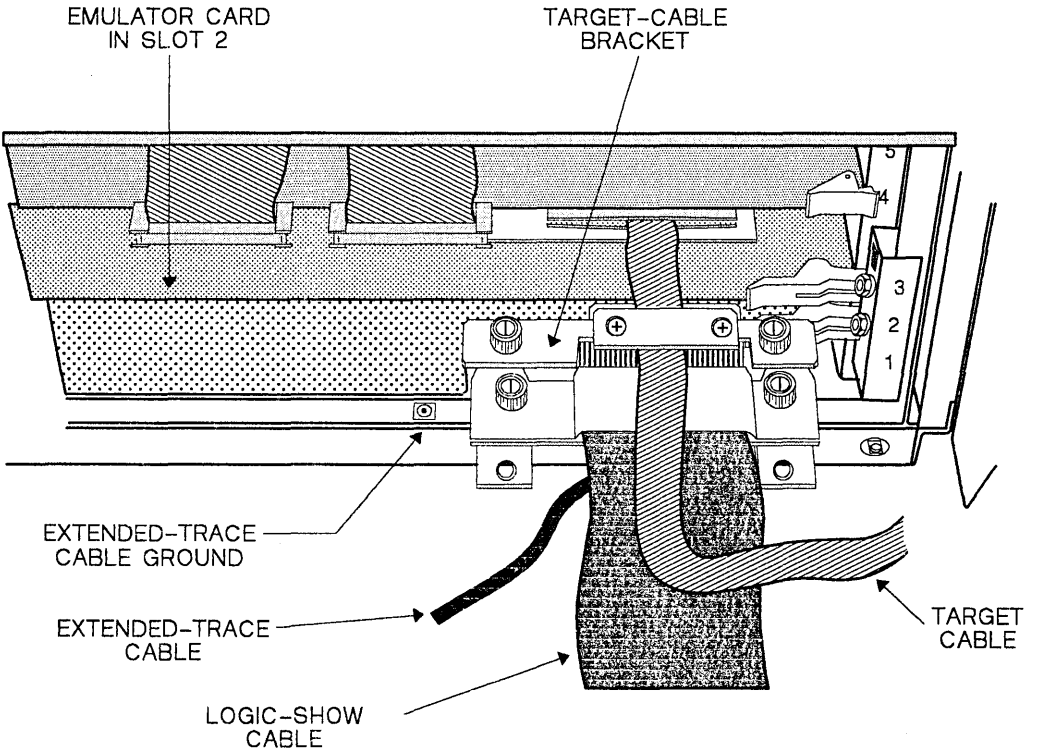


Figure 11-3. TMS370 XDS/22 Complete Chassis Configuration

11.1.3.1 Card Cage Cover Removal and Replacement

> **Caution:**
>
> **To avoid overheating and electromagnetic radiation, never operate an XDS/22 without its card cage cover in place.**

To remove the card cage cover:

1) Turn power OFF.

2) Loosen two captive thumb screws on lower front of XDS cabinet (Figure 11-4).

3) Apply slight upward pressure to cover. Pull bottom of cover up and away from chassis until bottom clears target-cable clamp hardware.

4) Let cover drop enough to clear top lip of chassis.

5) Remove cover. Be careful not to damage target-system cable.

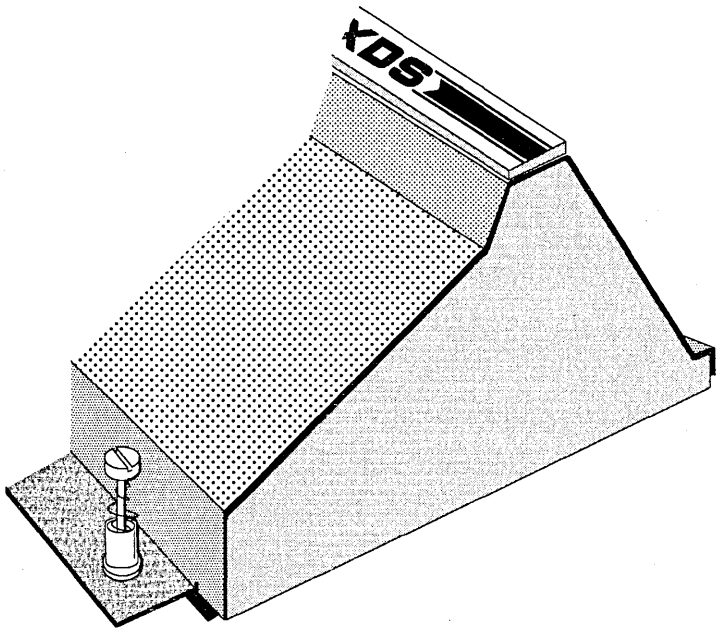


Figure 11-4. Card Cage Cover, Right Side

To replace the card cage cover:

1) Place top of cover under lip at top of XDS/22 unit.

2) Push cover bottom toward unit, ensuring that the two thumb screws start properly.

3) Tighten screws.

Caution:

Do not operate the XDS unit with the front panel removed. This panel must be installed for proper air circulation and cooling of the circuit boards. Also, the front panel is required to prevent leakage of RFI/EMI radiation.

11.1.4 Chassis Cable Mounting

The chassis cable mounting (Figure 11-5) anchors the LOGIC-SHOW cable strain-relief bracket. This application uses only the horizontal set of mounting holes.

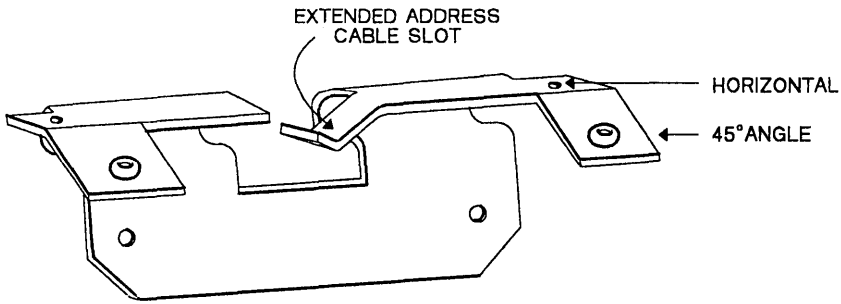


Figure 11-5. Chassis Cable Mounting

11.1.5 Power Supply

The switching power supply generates ±12 V and +5 V. DC power output current and tolerance figures are listed in Table 11-2 below.

All option boards that are installed must have their current requirements calculated into the total loading of the power supply.

Table 11-2. Maximum XDS/22 Chassis DC Power Available

DC Voltage	Tolerance	Current (Amps)
+5	±3%	20.0
+12	±2.5%	4.0
-12	±5%	2.0

Table 11-3 shows the DC power requirements for the XDS board set. This information can be used to calculate the total DC power requirements of the XDS system. When adding new boards to the system, this DC power calculation should be made to ensure that the XDS power supply will not be overloaded. Refer to Table 11-2 for the power supply rating information.

Table 11-3. XDS Board Set DC Power Requirements

Part Description	Board Part Number	+5 Volts (Amps)	+12 Volts (Amps)	-12 Volts (Amps)
TMS370 Emulator	2536810-1	3.0	0.0	0.0
BTT Card	2243660-2	4.5	0.0	0.0
Memory Exp/Comm	2311050-2	0.5	1.5	0.5

11.2 Memory Expansion/Communication Card

The Memory Expansion/Communications card (Figure 11-6) mounts processing logic for EIA-level signals to and from the XDS ports. It resides in slot number 4 in the chassis.

11.2.1 Card Removal

Remove the card from its slot by following this procedure:

1) Remove cables J41, J45, and J47 from front-edge card connectors.

2) Pull out on inside edge of ejector tabs to unseat card.

3) Carefully remove card from card cage.

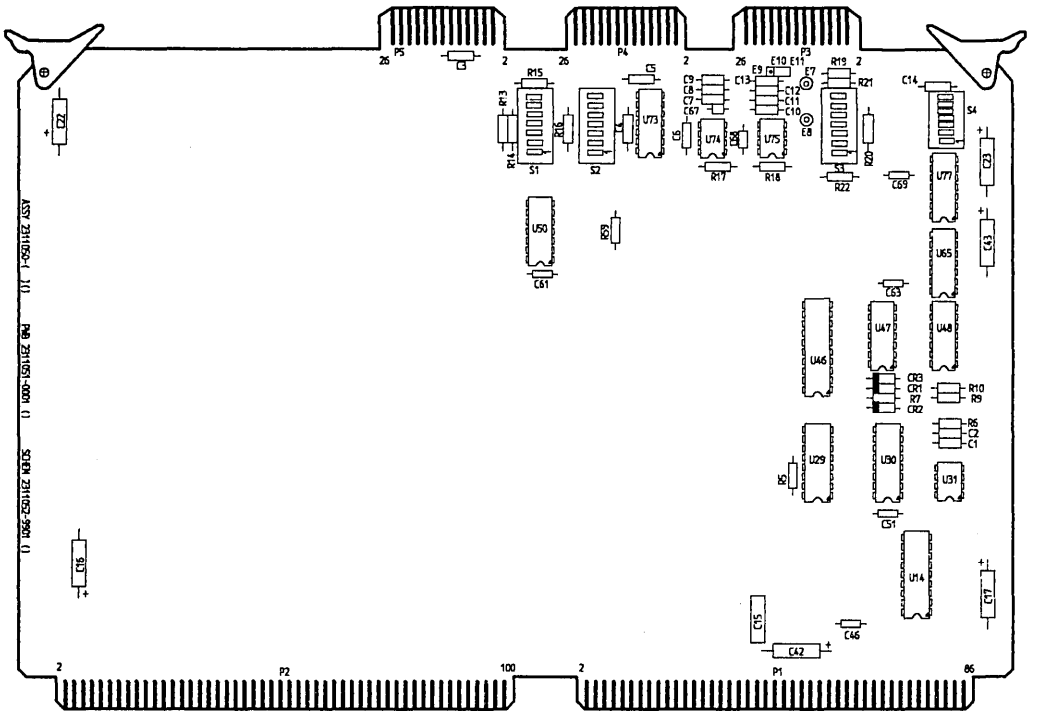


Figure 11-6. Memory Expansion/Communications Card

11.2.2 Switch Settings

Verify the settings of switches S3 and S4 on the Communications card and set correctly, if necessary. Refer to Table 11-4 for switch settings.

Table 11-4. Memory Expansion/Communications Card Switch Settings

Switch Position	S3	S4
1	off	off
2	off	off
3	off	off
4	on	off
5	on	on
6	on	on
7	on	off

11.2.3 Card Installation

1) Verify that the Card Assembly Revision level is correct on the chassis-configuration label inside the card cage cover.

2) Ensure proper switch settings before installing it in chassis slot #4, component side up. Press firmly on card edge or card-ejector tabs.

3) Connect J41, J45, and J47 (Figure 11-7).

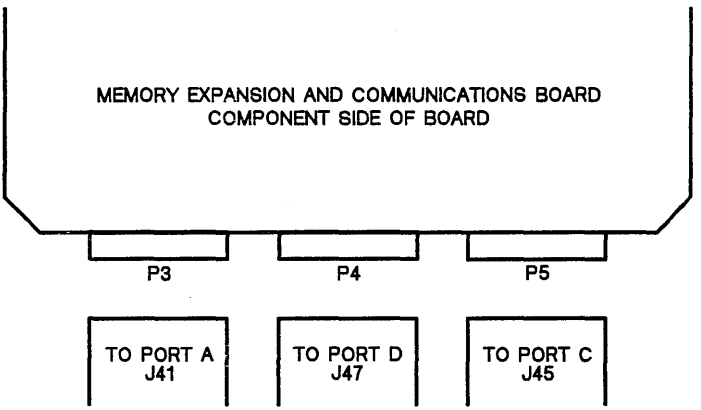


Figure 11-7. Communications Cable Connections

Note:

Do not attempt to connect J43.

11.3 The TMS370 Emulator Card

Figure 11-8 indicates the location of parts on the board that are of particular interest. To view these, remove the board according to the procedure given below.

11.3.1 Card Removal

> **Caution:**
>
> **Do not remove or install circuit boards or connectors with power applied to the chassis.**

This procedure is to be performed only at a static-free workstation. Adhere to normal static discharge handling procedures.

1) Remove power from the emulator and the target system.

2) Disconnect the target connector from the target system.

3) Protect the target connector leads by using a non-conductive protector.

4) Remove the emulator front panel by loosening the two thumb screws located at the bottom of the panel.

5) Detach the target cable strain relief by loosening the two thumb screws that anchor it to the chassis.

6) Locate the emulator board in slot number 2.

7) Using the two ejector tabs on the emulator board, remove the board from the chassis, keeping the target cable free of obstructions.

This section includes the following topics:

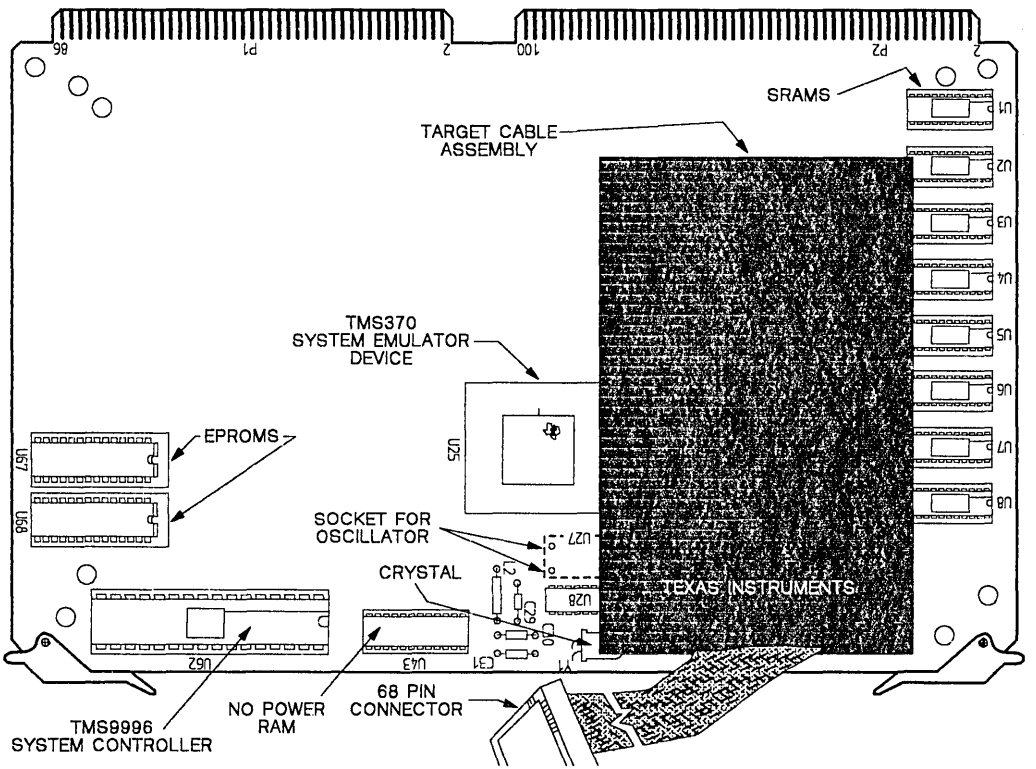


Figure 11-8. TMS370 Emulator Board with Target Cable Attached

11.3.2 Operating Frequency Options

The TMS370 emulator provides three methods of input for operating frequency. Referring to Figure 11-8 for locations, these methods are:

1) Onboard crystal oscillator package option at location U27.

2) Target system clock through the target cable assembly on the XTAL LOW/OSCIN pin. (Pin 31 on the 68-pin package or pin 5 on the 28-pin package.) Note that this option must be a TTL compatible clock; a crystal will not work.

3) Onboard crystal at location Y1 with Oscillator Tank Circuit network at locations L1 and C1 for 3rd Overtone crystals.

At emulator power-up, the previously selected clock (the selection is stored in non-volatile memory on the emulator board) is selected for the operating clock. If this clock does not properly operate the system, the next clock is used. If this fails, the third clock is used. The order for finding the proper

clock is as listed above starting with the selected clock and cycling through the list. Alternate clock selections are made through the appropriate initialization command.

The range of frequency inputs supported by the emulator matches that of the TMS370 microprocessor. Refer to the TMS370 Data Sheet for details.

11.3.3 Onboard Crystal Option

The onboard crystal at location Y1 may be replaced by removing it from the sockets contained on the emulator board. (Refer to Figure 11-8 for location.) The frequency of this crystal should not exceed the frequency range detailed in the TMS370 Data Sheet.

To ensure that 3rd Overtone crystals operate at the desired frequency, a Tank Circuit network is provided at locations L2 and C29. This provides a method of tuning the crystal input circuit. Inductor L2 and capacitor C29 should be chosen to disable the fundamental frequency of the crystal. The component values should satisfy the following equation:

$$Wo^2 = 1/(L2 \times C29),$$

where:

Wo is the fundamental radian frequency
L2 is the inductor value
C29 is the capacitor value

For example, to tune a 24 MHz 3rd overtone crystal:

Choose an inductor (L2) of 10 uH

Fundamental frequency = 24 MHz/3
= 8 MHz

Radian Frequency Wo = 2 x 3.1416 x 8 MHz

= 50.265 x 10^6 radians/sec

Capacitor value C29 = $1/(Wo^2 \times L2)$ = 39.6 pF

When installing components L2 and C29, remove capacitors C30 and C31 from the board. Refer to Figure 11-8 for locations.

11.3.4 Target System Clock Option

The clock from the target system may be used by driving the crystal low input lead on the target connector, pin 31 on the 68-pin package or pin 5 on the 28-pin package, with a TTL-compatible clock oscillator. The emulator supports the frequency range detailed in the TMS370 Data Sheet.

11.3.5 Onboard Crystal Oscillator Option

An onboard crystal oscillator package at location U27 (Figure 11-8) is also supported. The oscillator package must conform to the pinout in Table 11-5 for a 14-pin DIP.

Table 11-5. Crystal Oscillator Package Pinout

Pin Number	Function
1	N/C
7	Ground
8	Output
14	$+5V_{dc}$

Figure 11-9. Crystal Oscillator Pin Locations

11.3.6 Emulator Board Installation Procedure

Caution:

Do not remove or install circuit boards or connectors with power applied to the chassis.

This procedure is to be performed only at a static-free workstation. Adhere to normal static discharge handling procedures.

Make sure the Chassis Configuration Label found on the inside surface of the front cover contains the following information for the TMS370 Emulator board.

Table 11-6. Chassis Configuration Label Information

Block Title	Information Entered
PWB Description	TMS370
Part Number	2536810-1
Rev	Enter revision letter from TMS370 board

1) Ensure the emulator board is correctly configured for the desired board options listed previously in this section.

2) If the target cable is to be installed, connect the Plug Board sockets to the emulator header pins J4 through J7 (see Section 11.5.4).

3) Install card in XDS chassis slot #2, component side up. Seat card into backplane card connectors by pressing on the inside edge of card-ejector tabs. Be sure emulator target cable goes on top of both Breakpoint/Trace/Timing card LOGIC-SHOW and extended-address cables.

4) Secure cable-clamp thumb screws; do not overtighten.

5) Figure 11-10 shows the completely configured chassis.

6) See Section 11.1.3.1 for card cage cover replacement.

7) If the target cable assembly has been installed on the emulator board, attach the target cable strain relief to the chassis by tightening the two thumb screws.

8) Position the front panel by sliding the top of the panel under the chassis cover lip.

9) Tighten the two thumb screws located near the bottom of the front panel.

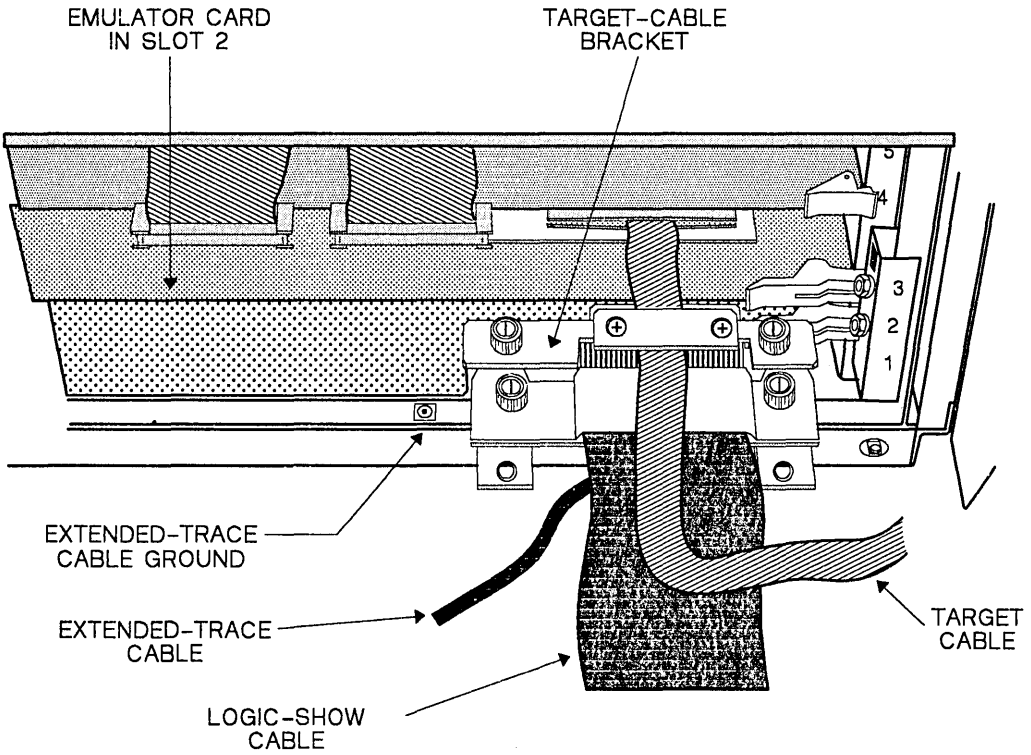


Figure 11-10. XDS/22 with Breakpoint/Trace and Emulator Cards Installed

11.4 Breakpoint/Trace/Timing Card

The Texas Instruments Breakpoint/Trace/Timing (BTT) Card gives you advanced breakpoint and tracing operation on up to 32 bits and also adds sequencing and trace time-stamping functions.

Address, data, and qualifier data from the XDS backplane passes through latches and lookup RAM to form 32-bit address/data comparisons and processor-cycle qualifications. These control event and delay counters, trace-sample enable, halt logic, sequence control, and timing logic.

The Breakpoint/Trace/Timing card LOGIC-SHOW cable also brings out signals for display on a logic analyzer or oscilloscope.

Jumper configuration and cabling connections for this card are shown in Figure 11-11.

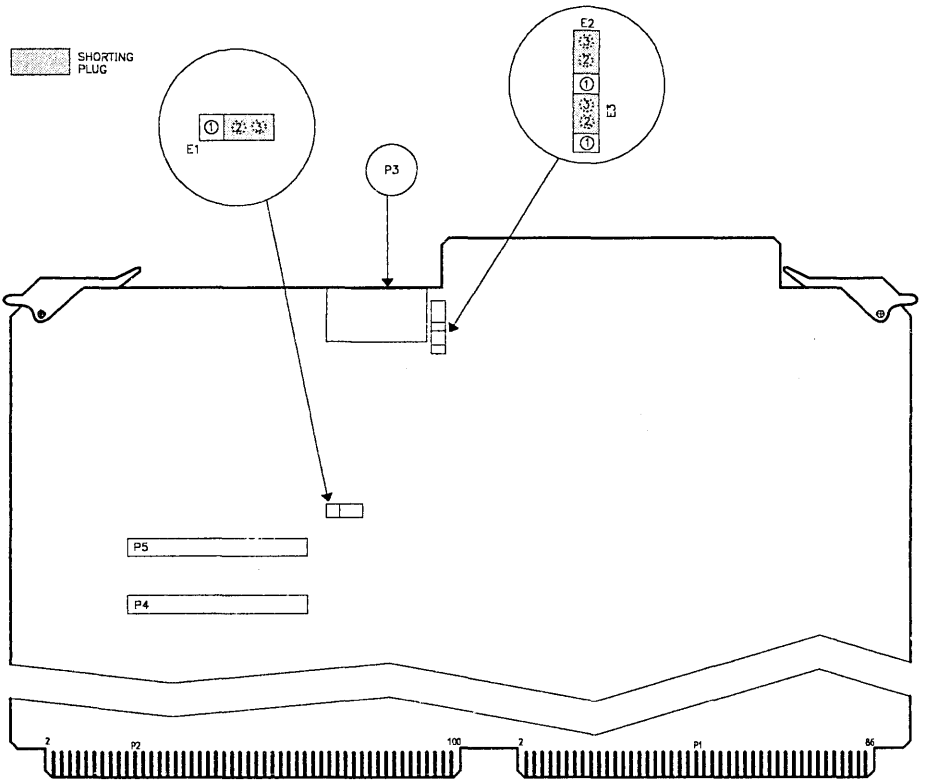


Figure 11-11. Breakpoint/Trace/Timing Card

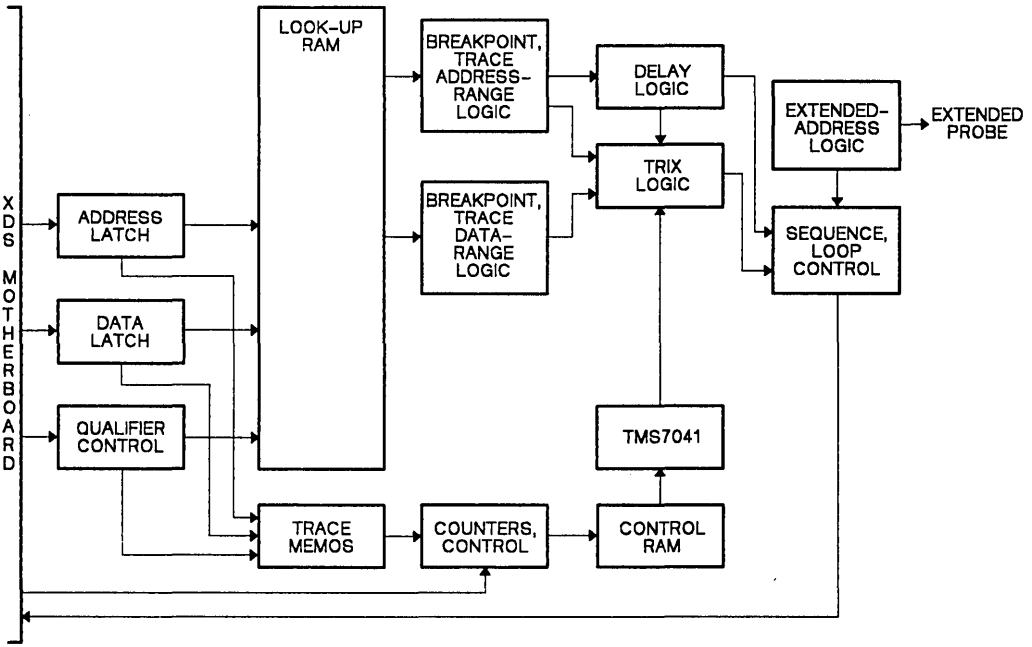


Figure 11-12. Breakpoint/Trace/Timing Card Block Diagram

11.4.1 Card Removal

Remove the Breakpoint/Trace/Timing card by following this procedure:

1) Remove the Emulator card according to the procedure outlined in Section 11.3.

2) Remove thumb screws holding LOGIC-SHOW cable clamp to chassis cable bracket.

3) Unplug extended-address cable and its ground connector.

4) Pull out on inside edge of each Breakpoint/Trace/Timing-card ejector tab simultaneously to unseat card.

5) Carefully remove card from card cage.

11.4.2 BTT Cabling

A Breakpoint/Trace/Timing card can have both a LOGIC-SHOW cable that plugs into P4 and P5 and an extended-address cable that plugs into P3.

11.4.2.1 LOGIC-SHOW Cable

The LOGIC-SHOW cable is flat, 2 inches (50mm) wide, and has a strain-relief bracket that fastens to the XDS chassis cable mount inside the XDS chassis (Figure 11-5 on Page 11-6). Figure 11-13 illustrates the LOGIC-SHOW cable with the ribbon cable connectors folded back toward the logic pod. These connectors are actually inside the chassis when the cable is installed.

Data, address, and qualifiers are brought out to the cable end to allow connection to a logic analyzer.

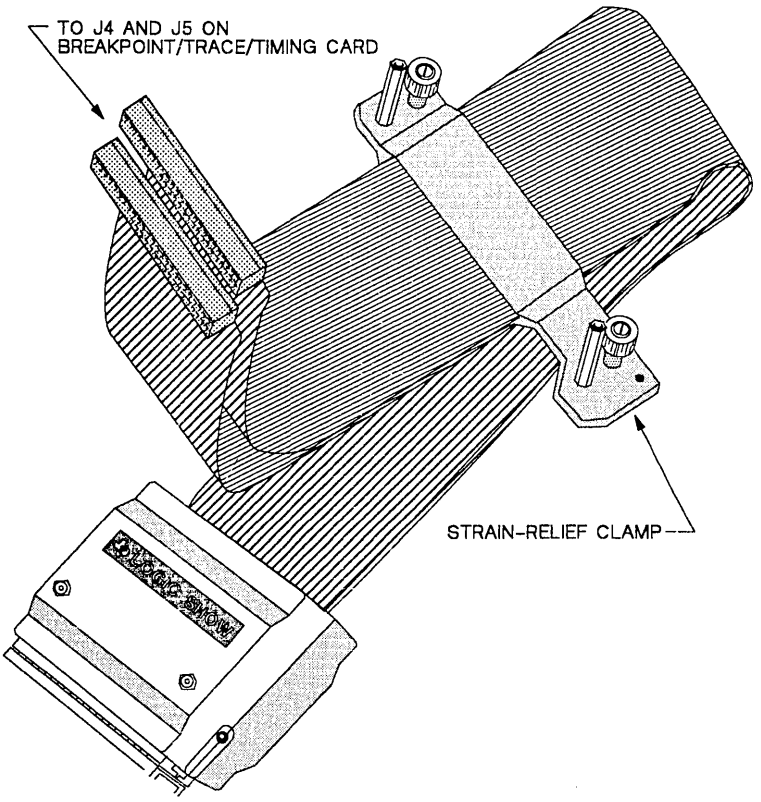


Figure 11-13. Texas Instruments LOGIC-SHOW Cable

The pinout of the logic pod is illustrated in Figure 11-14 and on the side of the logic pod opposite the LOGIC SHOW label.

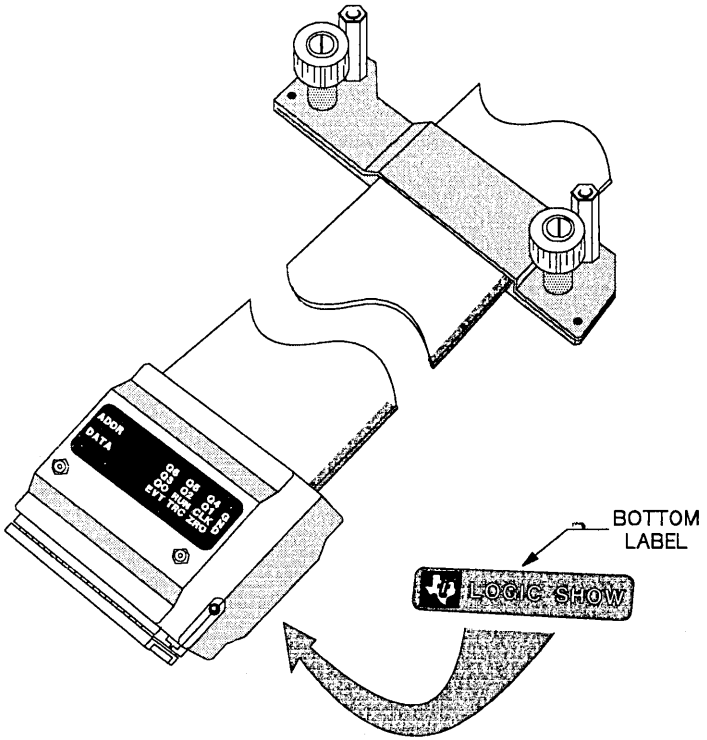


Figure 11-14. LOGIC-SHOW Cable Showing Pinout Label

Table 11-7. LOGIC-SHOW Cable Pinout

A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Q	Q	Q	G
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	0	0	0	N
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	6	5	4	D
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Q	Q	Q	G
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	N
5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	3	2	1	D
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	Q	R	C	G
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	0	U	L	N
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	0	N†K		D
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	E	T	Z	G
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	V	R	R	N
5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	T†C		0†D	

† indicates active-low signal.

In Table 11-7, the following definitions apply:

A00 - A15 Processor address lines
D08 - D15 Processor data lines

Q0 W/$\overline{R}$. Logic 0 indicates a READ cycle. Logic 1 indicates a WRITE cycle

Q1 $\overline{OCF}$ - Opcode fetch cycle. Logic 0 indicates that this cycle is an instruction opcode FETCH.

Q2 Not used.

Q3 $\overline{MEM}$. - Memory cycle. Active low should be used to qualify the CLOCK signal when memory cycle captures are desired.

Q4 Not used.

Q5 $\overline{INT}$ - INTERRUPT VECTOR FETCH. A Logic 0 indicates that these cycles were fetching an interrupt vector.

Q6 Not used.

$\overline{RUN}$ Emu running flag. Logic 0, emulator running; Logic 1, processor in command-entry mode. In Alternate-Run mode, emulator can still be running with $\overline{RUN}$ signal high.

CLK Sample clock. Low-high edge transition indicates valid address, data, and qualifier. Use to sample all signals on logic pod.

The $\overline{EVT}$, TRC, and $\overline{ZRO}$ signals occur one cycle after a valid $\overline{MEM}$ signal. They are results of BTT evaluation of the previous cycle due to BTT-card pipelining. Use these signals as additional clock qualifiers. The data and address are meaningless for that capture, but the capture signifies that the previous cycle had special meaning to the BTT card.

$\overline{ZRO}$ Breakpoint Halt-Pending Flag. Logic 0 indicates all breakpoint events have occurred and event count has decremented to zero. Breakpoint occurs after any delay count.

$\overline{EVT}$ Breakpoint Event flag. Logic 0 indicates hardware breakpoint event satisfied

TRC Trace-sample flag. Logic 1 indicates taking of trace sample.

11.4.2.2 Extended-Address Cable

The extended-address cable (Figure 11-15) is about 5/16 of an inch (8 mm) in diameter and fastens under the LOGIC-SHOW strain-relief bracket or adapter bracket in the semicircular groove in the chassis cable mount (see Section 11.1.4).

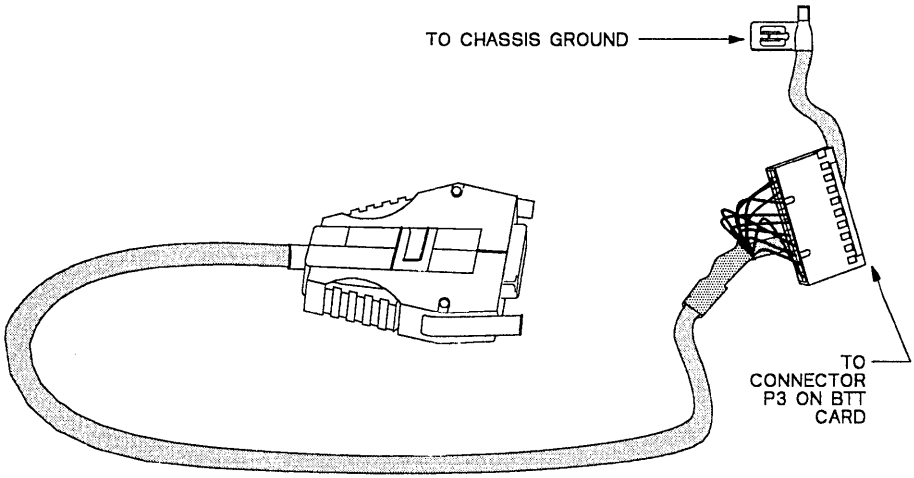


Figure 11-15. Extended-Address Cable

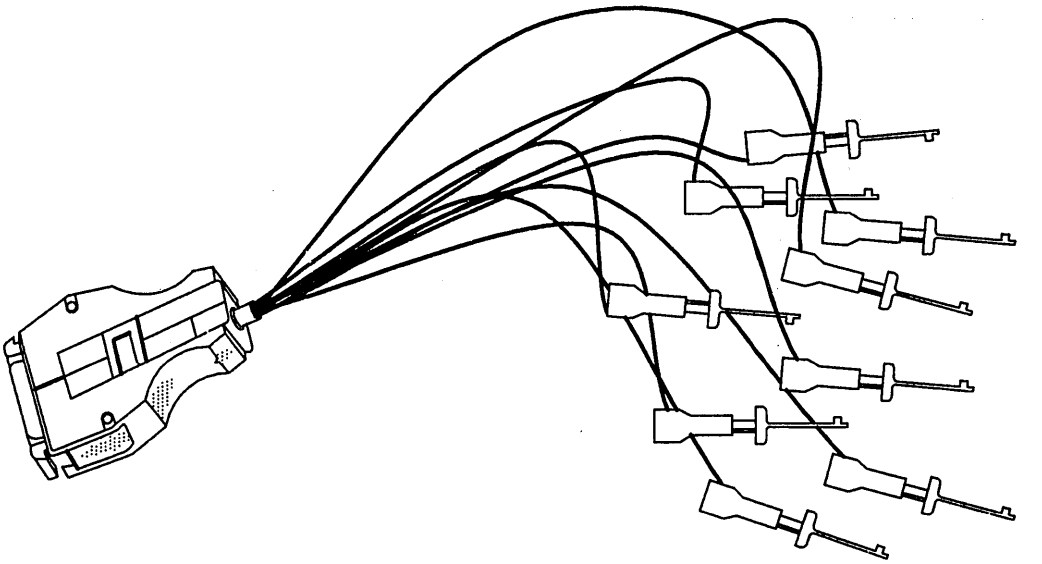


Figure 11-16. Extended-Address Cable Connector with Probes

11.4.3 Extended-Address Probes

These probes may be connected to additional address lines or to signals which will be qualifiers for Breakpoint or Trace applications. A color coding system (shown in Table 11-8) allows you to keep track of each line. Connect extended-address probes to the extended-address cable only when they are actually in use.

See the section on BTT command qualifiers (Section 8.2, page 8-4) for more information on utilizing these probes.

Table 11-8. Data Signal Values for Extended-Address Cable Lines

DATA BIT NUMBER	COLOR OF LINE
0 (LSB)	Black
1	Brown
2	Red
3	Orange
4	Yellow
5	Green
6	Blue
7 (MSB)	Violet
GROUND	White

11.4.4 Breakpoint/Trace/Timing Card Installation

1) If using a logic analyzer, connect cable J4 (stamped LOGIC SHOW) to BTT P4 and J5 to BTT P5, before installing the card. Be sure to align J4 with P4 and J5 with P5 and press connectors down firmly!

2) If using extended-address cable, plug its ground push-on connector into the XDS chassis clip.

3) Move card-ejector tabs parallel to side of card.

4) Place card in slot 1 (lowest) and push it into card cage until it reaches backplane card sockets, then back it out slightly until you can rotate outside edge of each ejector tab into small square hole on side of card cage (Figure 11-17). Take care to guide the extended-address cable past the card's soldered side.

> **Caution:**
>
> **Be careful. The projecting component leads are sharp.**

5) Press card into backplane card connectors by pressing on inside edge of card-ejector tabs.

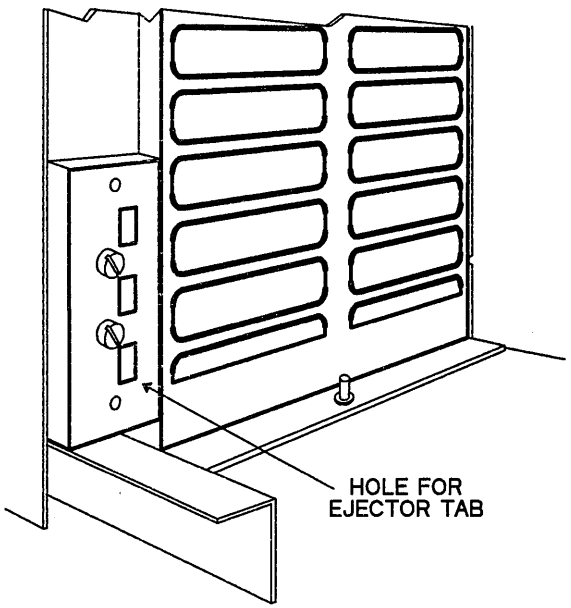


Figure 11-17. XDS/22 Card Cage Ejector Tab Holes

6) Plug extended-address cable into Breakpoint/Trace card connector P3 **after** card installation. If a logic analyzer is also used, run extended-address cable underneath LOGIC-SHOW cable.

7) Secure LOGIC-SHOW cable clamp on chassis cable-mount bracket with thumbscrews. Be sure extended-address cable goes in semicircular groove in chassis bracket.

8) If LOGIC-SHOW cable is not used, mount strain-relief adapter on chassis bracket mount by tightening the two thumb screws.

9) If installing no more cards, list Breakpoint/Trace/Timing card on the chassis-configuration label located on the card-cage cover.

After installation, replace cabinet cover as described on page 11-4.

11.5 Target Cables

> **Note:**
>
> The XDS may not operate properly when the target connector is connected to a target system that is not powered up.

11.5.1 Target Cable Assembly

Two target cable assemblies are provided to support emulation of both the TMS370C050 (68-pin PLCC) and the TMS370C010 (28-pin PLCC and DIP) devices. The target cable assemblies are shown in Figure 11-18 and Figure 11-19.

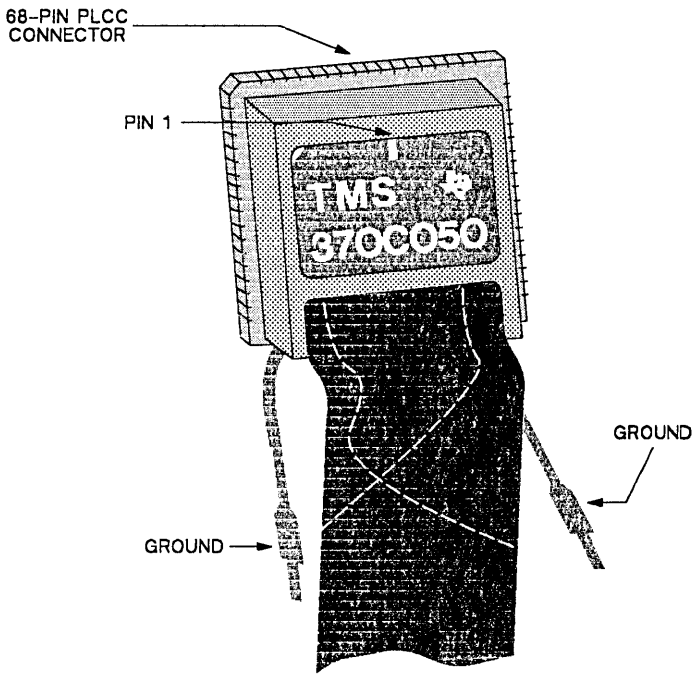


Figure 11-18. TMS370C050 Target Cable Assembly

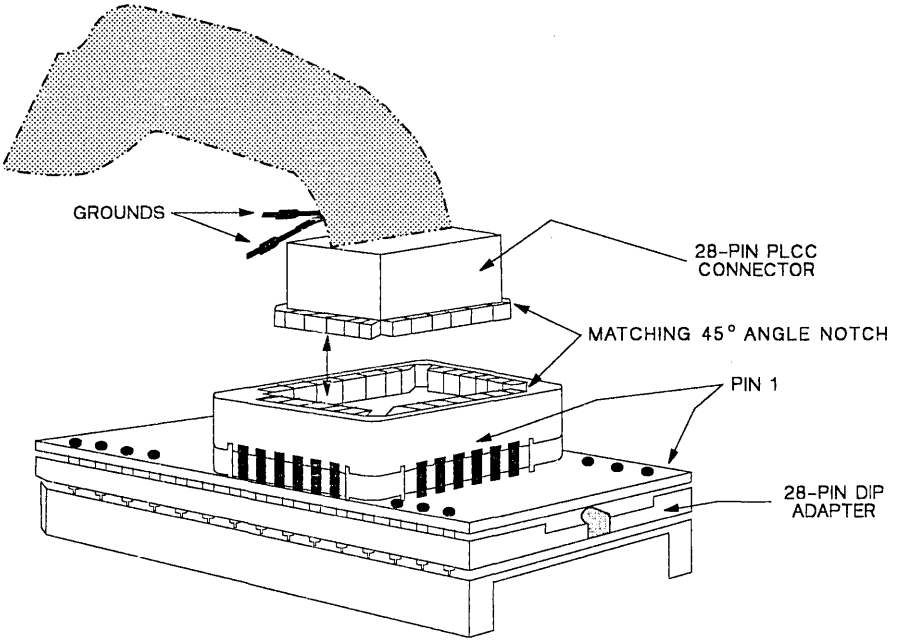


Figure 11-19. TMS370C010 Target Cable Assembly

11.5.2 Handling the Target Connector

> **Caution:**
>
> **Handle the target connector with extreme care at all times because the leads are fragile and can be easily damaged. When removing the PLCC from its socket, use a sharp instrument to pry the assembly from the target, as opposed to pulling it out by the attached cable.**

Use the following precautions when handling the target connector:

1) Protect against electrostatic discharge into the target connector, especially in extremely dry air conditions. An electrostatic discharge will damage the internal circuits in the XDS unit.

2) Power down the target system and the XDS unit when connecting or disconnecting the target connector to prevent an accidental short circuit.

3) When the target connector is not in use, the leads must be protected with a non-conductive protector to prevent damage to the leads and to prevent electrical shorts in the emulator. Conductive material should

never be used for protecting the target connector leads, since this can cause short circuits and damage to the emulator.

11.5.3 Connecting the Target Cable to the Target System

> **Caution:**
>
> **When installing the target connector into the target system, make sure that the lead orientation is correct.**

The target cable assembly terminates in either a 28-lead PLCC/DIP package or a 68-lead PLCC package at the target connector. Two external ground leads are provided on the target connector. These are located on the same side of the connector where the target cable exits. Use the black grounding clip provided to ensure that a good ground connection is made between the XDS unit and the target system.

For normal applications, plug the target connector into the target system in place of the microprocessor that is to be emulated. The target system can be any circuit that incorporates one or more processors. The target connector pinout for the TMS370 microprocessor and the signal characteristics are provided in the appropriate TMS370 Family device specification.

11.5.4 Installing the Target Cable Assembly in the Emulator

Two target cable assemblies are provided with the emulator: one to support the 28-lead TMS370C010 device and the other to support the 68-lead TMS370C050 device. To change from one cable assembly to the other, it is necessary to remove the emulator board from the XDS unit. Follow the emulator board removal procedure detailed in Section 11.3.1.

Figure 11-8, page 11-12, shows the target cable assembly connected to the TMS370 emulator board. At the emulator board, the target cable assembly terminates in a Plug Board, shown in Figure 11-20, which connects into headers J4 through J7 (34 pins each). The emulator board and target cables are marked for proper identification.

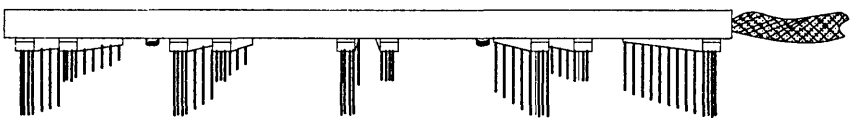


Figure 11-20. Target Cable Attachment

Installing the 68 or 28 pin target cable:

1) Remove the emulator board per Section 11.3.

2) Carefully remove the currently installed target cable by grasping the connectors of the target cable and pulling gently upward while holding the emulator card down on the table. Do not at any time grasp the cable itself or the strain relief attached to the cable and pull or twist.

3) Align the new target cable's connector with the emulator board according to Table 11-9. **Move the target cable slightly from side to side until all of the pins have slipped into the connectors.** Gently push the target cable down onto the emulator card until the cable is firmly attached. Verify that no pins are bent.

Table 11-9. Target Cable Connections

Emulator Connector	28 Pin PLCC Connector
J1	J1
J3	J4
J5	J2
J7	J3
J9	J5

Emulator Connector	68 Pin PLCC Connector
J2	J1
J4	J4
J6	J2
J8	J3
J11	J5

4) Replace the TMS370 emulator board per Section 11.3.

11.6 User Maintenance

> **Note:**
>
> Texas Instruments assumes no liability or contractual requirements by publication of maintenance schedules, procedures, and time intervals in this section; and intends these procedures only as a *guide* for qualified service personnel. **Procedures subject to change without notice.**

The only regular user maintenance is normal cleaning of the system cabinet and air filter.

11.6.1 Cleaning Cabinet

Texas Instruments recommends that you clean the cabinet monthly, or as necessary.

1) Turn power switch OFF and unplug line cord.
2) Wipe all unit surfaces with damp, lint-free cloth. For heavy dirt build-up, use mild soap solution. Avoid spraying soap solution on unit.
3) Clean air intake and exhaust areas.
4) Plug line cord back in.

11.6.2 Cleaning Air Filters

Texas Instruments recommends that you check the air filters monthly or more often, as necessary.

1) Turn power switch OFF and unplug unit from power source.
2) Remove the top cover:
   a) Set unit on its side and remove four 1/4" hexagonal-head screws marked in Figure 11-21 that hold top cover to cabinet base. Be sure to remove only the marked screws.
   b) Set unit back on its feet and remove four screws that hold top cover to rear panel.
   c) Remove top cover by spreading lower edges and lifting up.
3) Remove the air filter from each side of top cover by removing four retaining nuts. Lift each filter from mounting screws. Use a screwdriver (gently), if necessary.
4) Hold each filter up to window or overhead light. If either filter is very dirty, clean both as follows:
   a) Wash each filter in mild soap and water solution.
   b) Remove as much water as possible by blowing compressed air through filter or patting with paper towels.
   c) When completely dry, spray with water-soluble adhesive following directions on can label.

5) Reinstall the filters by following steps 1-3 in reverse.

Caution:

Filter must be completely dry before reinstallation.

Do NOT operate an XDS/22 without both air filters installed. In addition to blocking dust and dirt, these filters prevent accidental entrance of small objects through the ventilation slots.

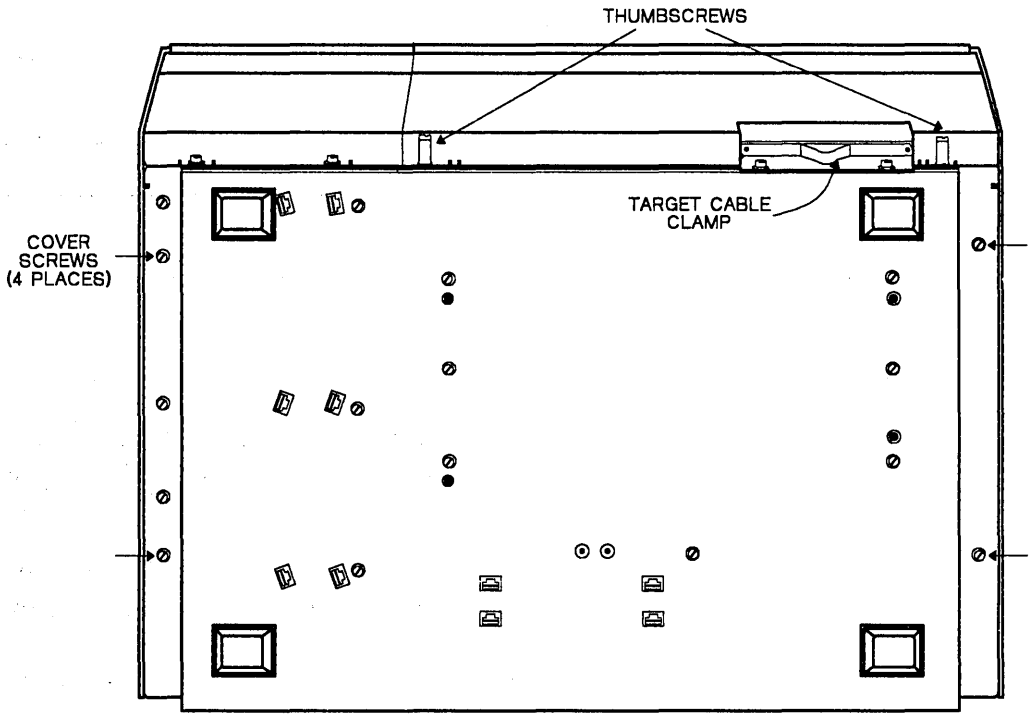


Figure 11-21. Top Cover Removal

11.6.3 Fuse Replacement

Only the AC line fuse on the XDS/22 rear panel is replaceable. Use this procedure:

1) Turn the XDS power switch OFF, then unplug the line cord from its receptacle.

2) Remove power cord from back of unit.

3) The power-cord connector also houses the fuse (Figure 11-2). Pry tab outward with small screwdriver.

4) Pull drawer unit out from power connector.

> **Warning:**
>
> **Do not connect XDS/22 to power source with fuse drawer pulled out.**

5) Lift narrow tab and pull fuse platform from drawer unit.

6) Replace fuse, if necessary, with appropriate selection from Table 11-10. Do **NOT** overfuse!

Table 11-10. Fuse Replacement Guide

XDS MODEL	FUSE RATING VOLTS	AMPS	TYPE	TI PART NUMBER	EQUIVALENT
2310990-1	125	3	SB	410822-60	Buss MDX-3
2310990-2	250	2	SB	2220531-4	Schurter 034.3120

7) Replace fuse platform in fuse drawer and push in until it locks in place.

8) Insert fuse drawer into power bulkhead until it is flush with the bulkhead.

9) Insert power cord in fuse holder/connector, then plug line cord into power receptacle.

10) Turn POWER switch on. If line fuse holds, proceed with normal operation.

11) If line fuse opens (fan not running), turn POWER switch off, then unseat all XDS cards.

12) Replace line fuse as previously described, reconnect power, then turn POWER switch on. If line fuse opens, turn POWER switch off, unplug the line cord, and refer to Section 11.7 to send equipment in for factory exchange or repair.

13) If line fuse holds, turn POWER switch off and reseat the emulator card.

14) Turn POWER switch on.

15) If line fuse opens, turn POWER switch off and send emulator card in for exchange or repair as detailed in Section 11.7.

16) If line fuse holds, repeat previous two steps, but reseat Memory/Communications card.

17) Repeat these steps for each remaining card. If any card causes the fuse to open, send that card in for exchange or repair as detailed in Section 11.7. After receiving the repaired or exchange card, don't forget to replace the fuse before attempting to operate the XDS.

If these procedures do not isolate the problem, refer to Section 11.7.

11.7 Factory Repair Information

Please consult the Factory Repair and Exchange Instructions that were enclosed with your development system for the most up-to-date instructions. The following section is included in case that document is unavailable and represents the most current information available at the time this book was printed.

The Texas Instruments Incorporated Microprocessor and Microcontroller Products Division (MMPD) Factory Repair Center at 12203 Southwest Freeway, Stafford, TX, 77477, M/S 730, (713) 274-2285, offers warranty repair or exchange at no charge[5] and non-warranty repair at standard labor and material rates for all current XDS products. Call the above number for price and delivery schedules.

11.7.1 Shipping Information

For any factory repair, do this:

1) Contact the Factory-Repair Center at (713) 274-2285 and ask for a Return Authorization Number and an exchange/repair questionnaire.

2) Fill out the questionnaire. Be sure to include the Return Authorization Number. Make a copy for your records.

3) Pack the XDS carefully and securely (preferably with the original packing material in the original shipping box) and send post or freight prepaid to:

   Texas Instruments Incorporated
   Microprocessor and Microcontroller Products Division
   Factory-Repair Center, M/S 730
   12203 Southwest Freeway
   Stafford, TX 77477

4) The package must contain the exchange/repair questionnaire completely filled out with the following information:

   - Return Authorization Number (Remember that Texas Instruments cannot accept returned equipment without this number)
   - Customer name, contact name, and telephone number
   - For warranty repair:
     a) Proof of date of purchase (required)
     b) Payment for return freight, if special return handling is required.
     c) Payment for expedited exchange, if that service requested.
   - For non-warranty repair, payment (if applicable), including expediting charges, if that service is requested.

---

[5] Except shipping.

- Model number (XDS/22) and serial number (from model plate on rear)
- "Ship-to" information, including address, amount of insurance, and shipping method. Texas Instruments ships UPS insured (minimum amount) unless you specify otherwise.
- Invoicing address, if applicable
- Request for return of same serial-numbered unit, if desired.
- Symptoms (please be as specific as possible) and type of service requested.

5) Make a copy of the waybill for your records.

Upon receipt, Texas Instruments inspects the equipment. If it does not meet the warranty conditions described on Page 11-36, Texas Instruments will notify you immediately.

11.7.2 Warranty Determination

Texas Instruments considers an XDS/22 to be in warranty if all these conditions apply:

1) You notify Texas Instruments of the problem within 90 days of purchase from Texas Instruments or an authorized distributor

2) The Factory-Repair Center receives the unit in standard Texas Instruments configuration with no customer additions

3) Factory-Repair Center inspection shows problem(s) not caused by accident, alteration, improper installation, improper testing, misuse, neglect, or unauthorized repair.

11.8 XDS/22 System Repair Guide

- If an XDS/22 does not power up when turned on, that is, the POWER light on the Operator Panel doesn't light and the fan doesn't run, refer to Section 11.8.1.
- If either the fan runs, but the POWER light doesn't light, or vice-versa, refer to Section 11.7.

> **Caution:**
>
> **Do not operate your XDS/22 if the fan doesn't work.**

- If the XDS/22 powers up, but the associated terminal does not display anything, refer to Section 11.8.2.
- If the XDS/22 and terminal work, but the target system doesn't respond, refer to Section 11.8.3.

11.8.1 XDS/22 Does Not Power Up

If an XDS/22 does not power up when turned on, that is, the POWER light on the Operator Panel doesn't light and the fan doesn't run, check the following:

1) Line cord defective or not connected to outlet or unit.
2) No power at outlet (check with voltmeter or plug known-good lamp or apparatus into outlet).
3) Chassis fuse blown (Refer to Section 11.6.3).

If none of these apply, refer to Section 11.7.

11.8.2 System Menu Not Displayed

If the XDS/22 powers up, but the computer doesn't display the XDS370 menu, follow this procedure:

1) If the computer screen displays nothing, follow normal troubleshooting procedures and/or local repair procedures for the computer.
2) If the screen displays a cursor, but doesn't display the XDS370 menu, do this:

   If the computer appears to be operating, first turn the computer off, then turn the XDS power switch off, then disconnect the terminal-XDS cable and ckeck its wiring against Table 2-3 (Emulator - Host Cable RS-232 Pin-outs).
3) If the cable is OK, or after repairing it, reconnect it and power the system up again.

4) If the problem remains, turn power off, remove the card-cage cover (page 11-4), then unseat Memory Expansion/Communications card in slot 4 by pulling out on inside edge of each card-ejector tab simultaneously, then reseat card by pressing on both card-ejector tabs simultaneously.

5) Remove and replace each of J41, J45, and J47.

6) Power the system up again. If the terminal doesn't display the system menu now, turn power off, then check settings of S3 and S4 on the Memory Expansion/Communications. (Refer to Table 11-4.)

7) If OK, or after resetting them, reseat the Memory Expansion/Communications, then power the system up again.

8) If the problem remains, turn power off, then unseat and reseat emulator card in slot 2.

9) Power the system up again. If the terminal doesn't display the system menu now, turn power off, then unseat the Breakpoint/Trace/Timing card *and leave it unseated.*

10) Power the system up again. If the terminal doesn't display the system menu now, turn power off, reseat the Breakpoint/Trace/Timing card, then power the system up again. If the menu still doesn't appear, replace the card-cage cover and refer to Section 11.7.

11.8.3 Target System Doesn't Respond

If the XDS/22 and terminal work, but the target system doesn't respond, follow this procedure:

1) Check target-system power.

2) If OK, power XDS and target system off, then remove target connector from system and check for broken or bent pins.

3) If OK, remove emulator card from XDS as follows and check that target-system cable is properly connected to card:

   a) Remove card-cage cover (Page 11-4).

   b) Loosen two captive screws holding target-system cable straining bracket.

   c) Unseat emulator card by pulling out on card-ejector tabs.

   d) Slide emulator card out of card cage.

   e) Carefully remove target cable from emulator card and check for bent or broken pins on card.

   f) If OK, carefully replace cable on card.

   g) Move card-ejector tabs parallel to card sides.

   h) Slide emulator card into card-cage slot 2 until card just touches sockets, then back card out slightly until you can rotate outside

edge of each ejector tab into small square hole on side of card cage.

i) Press card into backplane card connectors by pressing on card-ejector tabs.

If target system still does not respond, refer to Section 11.7, Factory Repair Information.

11.8.4 When All Else Fails

If the procedures in this Section do not restore proper system operation, contact the nearest Texas Instruments Regional Technology Center (RTC, numbers below).

Atlanta, GA	(Norcross, GA 30092)	(404) 662-7945
Boston, MA	(Waltham, MA 02154)	(617) 895-9196
Chicago, IL	(Arlington Heights, IL 60005)	(312) 640-2909
Dallas, TX	(Richardson, TX 75080)	(214) 680-5066
Irvine, CA	(Irvine, CA 92714)	(714) 660-8140

Also Available

Microcontroller Hotline	(713) 274-2370
Microcontroller Bulletin Board Baud 300, 1200, 2400; 8-bit data; no parity; one stop bit	(713) 274-3700

Appendix A

Debugger Command Overview

Top Level Commands	
Display	Fill the Display Window
Memory	Display any memory location
Pregs	Display peripheral registers
File	Display a file
Symbols	Display current module's symbols
m**O**dules	Display modules
Clear	Display system revisions
e**X**ecute	Run or Single-step the CPU
Instruction	Single-step instructions
Statement	Single-step statement (special for CALL)
Loop	Single-step once through a loop
While	Single-step while expression is true
Until	Single-step until expression is true
Function	Single-step until RTS or RTI is encountered
Go	Run from current PC
Reset	Software resest and run
t**A**rget reset/run	Wait for target to be reset, then run
Trace/Timer	Inspect trace samples
Register	Modify a register
Memory	Modify or fill memory
Modify	Modify a memory location
Fill	Fill a range of memory with a value
Assemble	Invoke symbolic line by line assembler
Breakpoint	Change code breakpoint settings
Add	Add a simple breakpoint
Delete	Delete a simple breakpoint
Remove all	Remove all simple breakpoints
Load	Load simple breakpoints from file
Save	Save simple breakpoints to a file
BTT	Enter BTT Programming Window (see below)
Evaluate	Evaluate an expression
Configure	Configure the debugger (see below)
Trace/Timer	Inspect trace samples
Position	Position trace screen at a specific sample
Top	Position trace screen at top of trace buffer
Bottom	Position trace screen at bottom of trace buffer
Lookup	Locate a specific trace sample
Format	Select the format for the Time Stamps
Save	Save trace buffer to a file
t**I**mers	Turn the timer window display on or off
e**X**ecute	Start the CPU
Load	Load a file
m**O**dule	Set the current module
Halt	Halt the CPU if running
System	Temporary escape to the operating system
Quit	Exit from the debugger

Configure Window Displayed Items	
Inspect or Change	
Run Mode	(run/continuous)
Trace Source	(memory/disassembly)
Clock Source	(target/osc/crystal)
Inspect Only	
Device Mode	(μprocessor/μcomputer)
Clock Period	
BTT Card Installed	(yes/no)
Configure Commands	
Memory Map	Configure the memory map
Load	Load memory map from file
Save	Save memory map to a file
Reset	Reset all memory ranges to device default
Device Select	Select a standard TMS370xx device
Reset	Reset the TMS370 device
Colors	Run the Color Setup utility (DBSETUP)
Breakpoint/BTT Subcommands	
Reset	Clear current configuration
Load	Load BTT configuration from a file
Save	Save BTT configuration to a file
Exit	Exit and program BTT with changes
Abort	Exit without programming the BTT

Appendix B

Debugger Error Messages

ADDRESS SHOULD BE 16 BYTE (BLOCK) BOUNDARY

- Starting addresses for memory map ranges must begin on a 16-byte boundary, of the form {nnn0h}.

BPT IS DISABLED ALREADY

- You tried to remove a breakpoint that is not set.

BPT IS ENABLED ALREADY

- You tried to set a breakpoint that is already set.

BPT IS NOT SET

- You tried to remove a breakpoint that is not set.

CAN'T CHANGE MODE, TOO MUCH ALLOCATED

- You have too many actions in a state to change the state mode from address-only to address+data.

CAN'T OPEN HELP FILE

- The file DBR370.HLP was not found in either the current directory or the directory pointed to by the DOS environment variable IPCDIR.

CAN'T PERFORM THE COMMAND, EMULATOR IS RUNNING

- You cannot execute a command that affects the emulator while it is running. See Section 6.1, Execute Command Options.

CAN'T UPDATE THE SCREEN WHILE EMULATOR IS RUNNING

- You cannot execute a command that affects the emulator while it is running. See Section 6.1, Execute Command Options.

COMMAND NOT RECOGNIZED -- HIT ESC KEY

- You tried to load and run a command file that contains invalid text.

DEBUGGER TERMINATING - INSUFFICIENT MEMORY

- You do not have enough free memory in your system to run the debugger.

DIVISION BY ZERO IS UNDEFINED

- You entered an expression which includes a division by zero.

EMULATOR NOT RESPONDING CORRECTLY - ABORT, RETRY, IGNORE?

- The debugger is not receiving signals from the emulator. See Figure 11-7, Communications Cable Connections.

END STATE MUST BE 0-3

- The only legal state numbers are 0, 1, 2, and 3.

ERROR WRITING FILE

- Caused by either a full disk or an attempt to use an illegal file name. Check your system.

EVENT COUNT MUST BE 0-0FFFFh

- The event count you entered on the BTT screen is out of legal range (0–65535 decimal).

ILLEGAL ADDRESS

- You entered an address that is out of range, or contains illegal characters.

ILLEGAL COMMAND LINE SWITCH

- You entered an invalid option on the debugger command line.

ILLEGAL OPERAND FOR THE UNARY '+' OR '-'

- The unary '+' or '-' operators are only valid for numeric data.

ILLEGAL OPERAND FOR THIS OPERATOR

- You entered an expression with an operator and operands of different types.

ILLEGAL OPERAND FOR UNARY 'NOT' OPERATOR

- The unary 'NOT' operator can only be applied to a numeric value.

INVALID MEMORY MAP ID

- You tried to load a memory map file that is in the wrong format; memory map files are stored in machine-readable form with a special id

INVALID MODULE NAME

- You tried to select a module that is not part of the object file, or no symbol table is loaded.

INVALID TIME VALUE

- You entered the time in an incorrect format, or out of range.

INVALID TRACE FORMAT OPTION

<filename> IS NOT A VALID <filetype> FILE

- The file you attempted to load (*filename*) is not in the correct format for a *filetype* file. *filetype* may be one of BTT, MMP, BP, etc.

JUMP DESTINATION MUST BE 0-3

The only states you can jump to are 0, 1, 2, and 3.

MISSING ASSIGNMENT OPERATOR

- The expression you entered requires an assignment operator.

MISSING COMMAND LINE SWITCH SETTING AFTER '='

- You omitted a parameter value in the debugger command line.

MISSING COMMAND LINE SWITCH BEFORE '='

- You omitted a keyword in the debugger command line.

MISSING OPERAND IN CONDITIONAL EXPRESSION

- The conditional expression you entered is missing one or both operands.

MISSING RELATIONAL OPERATOR

- The expression you entered requires a relational operator.

MISSING RELATIONAL OPERATOR IN CONDITIONAL EXPRESSION

- You entered an expression with no relational operator (< , > , = , etc.) where one is required.

MISSING RIGHT PARENTHESIS

- The expression you entered does not have matching left and right parentheses.

NO HELP FOR THAT CONTEXT

- The section of the debugger you are using does not have a help panel written specifically for it.

NO ROOM IN WINDOW FOR EXPRESSION

- The Expression Window is full. Delete some of the expressions and try again.

NO SYMBOL TABLE WAS LOADED
- The symbol table for the module you tried to make current (using the **O** command) is not in memory.

<x> NOT AVAILABLE FOR CLOCK

- The clock source you chose for the CPU is not operating properly.

NOT RUNNING

- You tried to halt the CPU (with the 'H' command) while the CPU was not running.

SYMBOL NOT FOUND IN SYMBOL TABLE

- You loaded a saved expression file without first loading the object file containing the necessary symbol file.

THIS MEMORY LOCATION IS NOT CONFIGURED

- You attempted to access a memory location that is not in the memory map.

UNABLE TO OPEN FILE <filename>
- The filename you specified cannot be found.

UNBALANCED QUOTES ON STRING

- The expression you entered does not have matching left and right quote marks.

UNDEFINED VARIABLE USED IN EXPRESSION

- You used a symbol that does not match either a register or a label defined in the code. Use 'Inspect Symbols' to look up the correct spelling and capitalization.

UNRECOGNIZED MEMORY MAP TYPE

- The id code in the .MMP file is not valid for a memory map file.

UNRECQGNIZED RADIX OPTION

- You used a radix specifier other than B, Q, D or H. See Section 3.4, Expressions.

Index

A

B

C

D

E

F

I

L

M

N

O

P

R

S

T

U

W

X

TI Worldwide Sales Offices

ALABAMA: Huntsville: 500 Wynn Drive, Suite 514, Huntsville, AL 35805, (205) 837-7530.

ARIZONA: Phoenix: 8825 N. 23rd Ave., Phoenix, AZ 85021, (602) 995-1007;**TUCSON:** 818 W. Miracle Mile, Suite 43, Tucson, AZ 85705, (602) 292-2640.

CALIFORNIA: Irvine: 17891 Cartwright Dr., Irvine, CA 92714, (714) 660-1200; **Roseville:** 1 Sierra Gate Plaza, Roseville, CA 95678, (916) 786-9208; **San Diego:** 4333 View Ridge Ave., Suite 100, San Diego, CA 92123, (619) 278-9601; **Santa Clara:** 5353 Betsy Ross Dr., Santa Clara, CA 95054, (408) 980-9000; **Torrance:** 690 Knox St., Torrance, CA 90502, (213) 217-7010; **Woodland Hills:** 21220 Erwin St., Woodland Hills, CA 91367, (818) 704-7759.

COLORADO: Aurora: 1400 S. Potomac Ave., Suite 101, Aurora, CO 80012, (303) 368-8000.

CONNECTICUT: Wallingford: 9 Barnes Industrial Park Rd., Barnes Industrial Park, Wallingford, CT 06492, (203) 269-0074.

FLORIDA: Altamonte Springs: 370 S. North Lake Blvd, Altamonte Springs, FL 32701, (305) 260-2116; **Ft. Lauderdale:** 2950 N.W. 62nd St., Ft. Lauderdale, FL 33309, (305) 973-8502; **Tampa:** 4803 George Rd., Suite 390, Tampa, FL 33634, (813) 885-7411.

GEORGIA: Norcross: 5515 Spalding Drive, Norcross, GA 30092, (404) 662-7900

ILLINOIS: Arlington Heights: 515 W. Algonquin, Arlington Heights, IL 60005, (312) 640-2925.

INDIANA: Ft. Wayne: 2020 Inwood Dr., Ft. Wayne, IN 46815, (219) 424-5174; **Carmel:** 550 Congressional Dr., Carmel, IN 46032, (317) 573-6400.

IOWA: Cedar Rapids: 373 Collins Rd. NE, Suite 201, Cedar Rapids, IA 52402, (319) 395-9550.

KANSAS: Overland Park: 7300 College Blvd., Lighton Plaza, Overland Park, KS 66210, (913) 451-4511.

MARYLAND: Columbia: 8815 Centre Park Dr., Columbia MD 21045, (301) 964-2003.

MASSACHUSETTS: Waltham: 950 Winter St., Waltham, MA 02154, (617) 895-9100.

MICHIGAN: Farmington Hills: 33737 W. 12 Mile Rd., Farmington Hills, MI 48018, (313) 553-1569. **Grand Rapids:** 3075 Orchard Vista Dr. S.E., Grand Rapids, MI 49506, (616) 957-4200.

MINNESOTA: Eden Prairie: 11000 W. 78th St., Eden Prairie, MN 55344 (612) 828-9300.

MISSOURI: St. Louis: 11816 Borman Drive, St. Louis, MO 63146, (314) 569-7600.

NEW JERSEY: Iselin: 485E U.S. Route 1 South, Parkway Towers, Iselin, NJ 08830 (201) 750-1050.

NEW MEXICO: Albuquerque: 2820-D Broadbent Pkwy NE, Albuquerque, NM 87107, (505) 345-2555.

NEW YORK: East Syracuse: 6365 Collamer Dr., East Syracuse, NY 13057, (315) 463-9291; **Melville:** 1895 Walt Whitman Rd., P.O. Box 2936, Melville, NY 11747, (516) 454-6600; **Pittsford:** 2851 Clover St., Pittsford, NY 14534, (716) 385-6770; **Poughkeepsie:** 385 South Rd., Poughkeepsie, NY 12601, (914) 473-2900.

NORTH CAROLINA: Charlotte: 8 Woodlawn Green, Woodlawn Rd., Charlotte, NC 28210, (704) 527-0933; **Raleigh:** 2809 Highwoods Blvd., Suite 100, Raleigh, NC 27625, (919) 876-2725.

OHIO: Beachwood: 23775 Commerce Park Rd., Beachwood, OH 44122, (216) 464-6100; **Beavercreek:** 4200 Colonel Glenn Hwy., Beavercreek, OH 45431, (513) 427-6200.

OREGON: Beaverton: 6700 SW 105th St., Suite 110, Beaverton, OR 97005, (503) 643-6758.

PENNSYLVANIA: Blue Bell: 670 Sentry Pkwy, Blue Bell, PA 19422, (215) 825-9500.

PUERTO RICO: Hato Rey: Mercantil Plaza Bldg., Suite 505, Hato Rey, PR 00918, (809) 753-8700.

TENNESSEE: Johnson City: Erwin Hwy, P.O. Drawer 1255, Johnson City, TN 37605 (615) 461-2192.

TEXAS: Austin: 12501 Research Blvd., Austin, TX 78759, (512) 250-7655; **Richardson:** 1001 E. Campbell Rd., Richardson, TX 75081, (214) 680-5082; **Houston:** 9100 Southwest Frwy., Suite 250, Houston, TX 77074, (713) 778-6592; **San Antonio:** 1000 Central Parkway South, San Antonio, TX 78232, (512) 496-1779.

UTAH: Murray: 5201 South Green St., Suite 200, Murray, UT 84123, (801) 266-8972.

WASHINGTON: Redmond: 5010 148th NE, Bldg B, Suite 107, Redmond, WA 98052, (206) 881-3080.

WISCONSIN: Brookfield: 450 N. Sunny Slope, Suite 150, Brookfield, WI 53005, (414) 782-2899.

CANADA: Nepean: 301 Moodie Drive, Mallorn Center, Nepean, Ontario, Canada, K2H9C4, (613) 726-1970. **Richmond Hill:** 280 Centre St. E., Richmond Hill L4C1B1, Ontario, Canada (416) 884-9181; **St. Laurent:** Ville St. Laurent Quebec, 9460 Trans Canada Hwy., St. Laurent, Quebec, Canada H4S1R7, (514) 336-1860.

ARGENTINA: Texas Instruments Argentina Viamonte 1119, 1053 Capital Federal, Buenos Aires, Argentina, 541/748-3699

AUSTRALIA (& NEW ZEALAND): Texas Instruments Australia Ltd.: 6-10 Talavera Rd., North Ryde (Sydney), New South Wales, Australia 2113, 2 + 887-1122; 5th Floor, 418 St. Kilda Road, Melbourne, Victoria, Australia 3004, 3 + 267-4677; 171 Philip Highway, Elizabeth, South Australia 5112, 8 + 255-2066.

AUSTRIA: Texas Instruments Ges.m.b.H.: Industriestrabe B/16, A-2345 Brunn/Gebirge, 2236-846210.

BELGIUM: Texas Instruments N.V. Belgium S.A.: 11, Avenue Jules Bondetlaan 11, 1140 Brussels, Belgium, (02) 242-3080.

BRAZIL: Texas Instruments Electronicos do Brasil Ltda.: Rua Paes Leme, 524-7 Andar Pinheiros, 05424 Sao Paulo, Brazil, 0815-6166.

DENMARK: Texas Instruments A/S, Mairelundvej 46E, 2730 Herlev, Denmark, 2 - 91 74 00.

FINLAND: Texas Instruments Finland OY: Ahertajantie 3, P.O. Box 81, ESPOO, Finland, (90) 0-461-422.

FRANCE: Texas Instruments France: Paris Office, BP 67 8-10 Avenue Morane-Saulnier, 78141 Velizy-Villacoublay cedex (1) 30 70 1003.

GERMANY (Fed. Republic of Germany): Texas Instruments Deutschland GmbH: Haggertystrasse 1, 8050 Freising, 8161 + 80-4591; Kurfuerstendamm 195/196, 1000 Berlin 15, 30 + 882-7365; III, Hagen 43/Kibbelstrasse, .19, 4300 Essen, 201-24250; Kirchhorsterstrasse 2, 3000 Hannover 51, 511 + 648021; Maybachstrabe 11, 7302 Ostfildern 2-Nelingen, 711 + 34030.

TEXAS INSTRUMENTS

HONG KONG: Texas Instruments Hong Kong Ltd., 8th Floor, World Shipping Ctr., 7 Canton Rd., Kowloon, Hong Kong, (852) 3-7351223.

IRELAND: Texas Instruments (Ireland) Limited: 7/8 Harcourt Street, Stillorgan, County Dublin, Eire, 1 781677.

ITALY: Texas Instruments Italia S.p.A. Divisione Semiconduttori: Viale Europa, 40, 20093 Cologne Monzese (Milano), (02) 253001; Via Castello della Magliana, 38, 00148 Roma, (06) 5222651; Via Amendola, 17, 40100 Bologna, (051) 554004.

JAPAN: Tokyo Marketing/Sales (Headquarters): Texas Instruments Japan Ltd., MS Shibaura Bldg., 9F, 4-13-23 Shibaura, Minato-ku, Tokyo 108, Japan, 03-769-8700. Texas Instruments Japan Ltd.: Nissho-Iwai Bldg. 5F, 30 Imabashi 3-chome, Higashi-ku, Osaka 541, Japan, 06-294-1881; Daini Toyota West Bldg. 7F, 10-27 Meieki 4-chome, Nakamura-ku, Nagoya 450, 052-583-8691; Daiichi Seimei Bldg. 6F, 3-10 Oyama-cho, Kanazawa 920, Ishikawa-ken, 0762-23-5471; Daiichi Olympic Tachikawa Bldg. 6F, 1-25-12 Akebono-cho, Tachikawa 190, Tokyo, 0425-27-6426; Matsumoto Showa Bldg. 6F, 2-11 Fukashi 1-chome, Matsumoto 390, Nagano-ken, 0263-33-1060; Yokohama Nishiguchi KN Bldg. 6F, 2-8-4 Kita-Saiwai-cho, Nishi-ku, Yokohama 220, 045-322-6741; Nihon Seimei Kyoto Yasaka Bldg. 5F, 843-2 Higashi Shiokohjidori, Nishinotoh-in Higashi-iru, Shiokouji, Shimogyo-ku, Kyoto 600, 075-341-7713; 2597-1, Aza Harudai, Oaza Yasaka, Kitsuki 873, Oita-ken, 09786-3-3211; Miho Plant, 2350 Kihara Miho-mura, Inashiki-gun 300-04, Ibaragi-ken, 0298-85-2541.

KOREA: Texas Instruments Korea Ltd., 28th Fl., Trade Tower, #159, Samsung-Dong, Kangnam-ku, Seoul, Korea 2 + 551-2810.

MEXICO: Texas Instruments de Mexico S.A.: Alfonso Reyes—115, Col. Hipodromo Condesa, Mexico, D.F., Mexico 06120, 525/525-3860.

MIDDLE EAST: Texas Instruments: No. 13, 1st Floor Mannai Bldg., Diplomatic Area, P.O. Box 26335, Manama Bahrain, Arabian Gulf, 973 + 274681.

NETHERLANDS: Texas Instruments Holland B.V., 19 Hogehilweg, 1100 AZ Amsterdam—Zuidoost, Holland 20 + 5602911.

NORWAY: Texas Instruments Norway A/S: PB106, Refstad 0585, Oslo 5, Norway, (2) 155090.

PEOPLES REPUBLIC OF CHINA: Texas Instruments China Inc., Beijing Representative Office, 7-05 Citic Bldg., 19 Jianguomenwai Dajje, Beijing, China, (861) 5002255, Ext. 3750.

PHILIPPINES: Texas Instruments Asia Ltd.: 14th Floor, Ba- Lepanto Bldg., Paseo de Roxas, Makati, Metro Manila, Philippines, 817-60-31.

PORTUGAL: Texas Instruments Equipamento Electronico (Portugal), Lda.: Rua Eng. Frederico Ulrich, 2650 Moreira Da Maia, 4470 Maia, Portugal, 2-948-1003.

SINGAPORE (+ INDIA, INDONESIA, MALAYSIA, THAILAND): Texas Instruments Singapore (PTE) Ltd., Asia Pacific Division, 101 Thompson Rd. #23-01, United Square, Singapore 1130, 350-8100.

SPAIN: Texas Instruments Espana, S.A.: C/Jose Lazaro Galdiano No. 6, Madrid 28036, 1/458.14.58.

SWEDEN: Texas Instruments International Trade Corporation (Sverigefilialen): S-164-93, Stockholm, Sweden, 8 - 752-5800.

SWITZERLAND: Texas Instruments, Inc., Reidstrasse 6, CH-8953 Dietikon (Zuerich) Switzerland, 1-740 2220.

TAIWAN: Texas Instruments Supply Co., 9th Floor Bank Tower, 205 Tun Hwa N. Rd., Taipei, Taiwan, Republic of China, 2 + 713-9311.

UNITED KINGDOM: Texas Instruments Limited: Manton Lane, Bedford, MK41 7PA, England, 0234 270111.

TI Sales Offices

ALABAMA: Huntsville (205) 837-7530.

ARIZONA: Phoenix (602) 995-1007;
Tucson (602) 292-2640.

CALIFORNIA: Irvine (714) 660-1200;
Roseville (916) 786-9208;
San Diego (619) 278-9601;
Santa Clara (408) 980-9000;
Torrance (213) 217-7010;
Woodland Hills (818) 704-7759.

COLORADO: Aurora (303) 368-8000.

CONNECTICUT: Wallingford (203) 269-0074.

FLORIDA: Altamonte Springs (305) 260-2116;
Ft. Lauderdale (305) 973-8502;
Tampa (813) 885-7411.

GEORGIA: Norcross (404) 662-7900.

ILLINOIS: Arlington Heights (312) 640-2925.

INDIANA: Carmel (317) 573-6400;
Ft. Wayne (219) 424-5174.

IOWA: Cedar Rapids (319) 395-9550.

KANSAS: Overland Park (913) 451-4511.

MARYLAND: Columbia (301) 964-2003.

MASSACHUSETTS: Waltham (617) 895-9100.

MICHIGAN: Farmington Hills (313) 553-1569;
Grand Rapids (616) 957-4200.

MINNESOTA: Eden Prairie (612) 828-9300.

MISSOURI: St. Louis (314) 569-7600.

NEW JERSEY: Iselin (201) 750-1050.

NEW MEXICO: Albuquerque (505) 345-2555.

NEW YORK: East Syracuse (315) 463-9291;
Melville (516) 454-6600;
Pittsford (716) 385-6770;
Poughkeepsie (914) 473-2900.

NORTH CAROLINA: Charlotte (704) 527-0933;
Raleigh (919) 876-2725.

OHIO: Beachwood (216) 464-6100;
Beaver Creek (513) 427-6200.

OREGON: Beaverton (503) 643-6758.

PENNSYLVANIA: Blue Bell (215) 825-9500.

PUERTO RICO: Hato Rey (809) 753-8700.

TENNESSEE: Johnson City (615) 461-2192.

TEXAS: Austin (512) 250-7655;
Houston (713) 778-6592;
Richardson (214) 680-5082;
San Antonio (512) 496-1779.

UTAH: Murray (801) 266-8972.

WASHINGTON: Redmond (206) 881-3080.

WISCONSIN: Brookfield (414) 782-2899.

CANADA: Nepean, Ontario (613) 726-1970;
Richmond Hill, Ontario (416) 884-9181;
St. Laurent, Quebec (514) 336-1860.

TI Regional Technology Centers

CALIFORNIA: Irvine (714) 660-8105;
Santa Clara (408) 748-2220;

GEORGIA: Norcross (404) 662-7945.

ILLINOIS Arlington Heights (312) 640-2909.

MASSACHUSETTS: Waltham (617) 895-9196.

TEXAS: Richardson (214) 680-5066.

CANADA: Nepean, Ontario (613) 726-1970.

TI Distributors

TI AUTHORIZED DISTRIBUTORS
Arrow/Kierulff Electronics Group
Arrow (Canada)
Future Electronics (Canada)
GRS Electronics Co., Inc.
Hall-Mark Electronics
Marshall Industries
Newark Electronics
Schweber Electronics
Time Electronics
Wyle Laboratories
Zeus Components

–OBSOLETE PRODUCT ONLY–
Rochester Electronics, Inc.
Newburyport, Massachusetts
(508) 462-9332

ALABAMA: Arrow/Kierulff (205) 837-6955;
Hall-Mark (205) 837-8700; Marshall (205) 881-9235;
Schweber (205) 895-0480.

ARIZONA: Arrow/Kierulff (602) 437-0750;
Hall-Mark (602) 437-1200; Marshall (602) 496-0290;
Schweber (602) 431-0030; Wyle (602) 866-2888.

CALIFORNIA: Los Angeles/Orange County:
Arrow/Kierulff (818) 701-7500, (714) 838-5422;
Hall-Mark (818) 773-4500, (714) 669-4100;
Marshall (818) 407-0101, (818) 459-5500,
(714) 458-5395; Schweber (818) 880-9686;
(714) 863-0200, (213) 320-8090; Wyle (818) 880-9000,
(714) 863-9953; Zeus (714) 921-9000; (818) 889-3838;
Sacramento: Hall-Mark (916) 624-9781;
Marshall (916) 635-9700; Schweber (916) 364-0222;
Wyle (916) 638-5282;
San Diego: Arrow/Kierulff (619) 565-4800;
Hall-Mark (619) 268-1201; Marshall (619) 578-9600;
Schweber (619) 450-0454; Wyle (619) 565-9171;
San Francisco Bay Area: Arrow/Kierulff (408) 745-6600,
Hall-Mark (408) 432-0900; Marshall (408) 942-4600;
Schweber (408) 432-7171; Wyle (408) 727-2500;
Zeus (408) 998-5121.

COLORADO: Arrow/Kierulff (303) 790-4444;
Hall-Mark (303) 790-1662; Marshall (303) 451-8383;
Schweber (303) 799-0258; Wyle (303) 457-9953.

CONNETICUT: Arrow/Kierulff (203) 265-7741;
Hall-Mark (203) 271-2844; Marshall (203) 265-3822;
Schweber (203) 264-4700.

FLORIDA: Ft. Lauderdale:
Arrow/Kierulff (305) 429-8200; Hall-Mark (305) 971-9280;
Marshall (305) 977-4880; Schweber (305) 977-7511;
Orlando: Arrow/Kierulff (407) 323-0252;
Hall-Mark (407) 830-5855; Marshall (407) 767-8585;
Schweber (407) 331-7555; Zeus (407) 365-3000;
Tampa: Hall-Mark (813) 530-4543;
Marshall (813) 576-1399; Schweber (813) 541-5100.

GEORGIA: Arrow/Kierulff (404) 449-8252;
Hall-Mark (404) 447-8000; Marshall (404) 923-5750;
Schweber (404) 449-9170.

ILLINOIS: Arrow/Kierulff (312) 250-0500;
Hall-Mark (312) 860-3800; Marshall (312) 490-0155;
Newark (312) 784-5100; Schweber (312) 364-3750.

INDIANA: Indianapolis: Arrow/Kierulff (317) 243-9353;
Hall-Mark (317) 872-8875; Marshall (317) 297-0483;
Schweber (317) 843-1050.

IOWA: Arrow/Kierulff (319) 395-7230;
Schweber (319) 373-1417.

KANSAS: Kansas City: Arrow/Kierulff (913) 541-9542;
Hall-Mark (913) 888-4747; Marshall (913) 492-3121;
Schweber (913) 492-2922.

MARYLAND: Arrow/Kierulff (301) 995-6002;
Hall-Mark (301) 988-9800; Marshall (301) 235-9464;
Schweber (301) 840-5900; Zeus (301) 997-1118.

MASSACHUSETTS Arrow/Kierulff (508) 658-0900;
Hall-Mark (508) 667-0902; Marshall (508) 658-0810;
Schweber (617) 275-5100; Time (617) 532-6200;
Wyle (617) 273-7300; Zeus (617) 863-8800.

MICHIGAN: Detroit: Arrow/Kierulff (313) 462-2290;
Hall-Mark (313) 462-1205; Marshall (313) 525-5850;
Newark (313) 967-0600; Schweber (313) 525-8100;
Grand Rapids: Arrow/Kierulff (616) 243-0912.

MINNESOTA: Arrow/Kierulff (612) 830-1800;
Hall-Mark (612) 941-2600; Marshall (612) 559-2211;
Schweber (612) 941-5280.

MISSOURI: St. Louis: Arrow/Kierulff (314) 567-6888;
Hall-Mark (314) 291-5350; Marshall (314) 291-4650;
Schweber (314) 739-0526.

NEW HAMPSHIRE: Arrow/Kierulff (603) 668-6968;
Schweber (603) 625-2250.

NEW JERSEY: Arrow/Kierulff (201) 538-0900,
(609) 596-8000; GRS Electronics (609) 964-8560;
Hall-Mark (201) 575-4415, (201) 882-9773,
(609) 235-1900; Marshall (201) 882-0320,
(609) 234-9100; Schweber (201) 227-7880.

NEW MEXICO: Arrow/Kierulff (505) 243-4566.

NEW YORK: Long Island:
Arrow/Kierulff (516) 231-1000; Hall-Mark (516) 737-0600;
Marshall (516) 273-2424; Schweber (516) 334-7474;
Zeus (914) 937-7400;
Rochester: Arrow/Kierulff (716) 427-0300;
Hall-Mark (716) 425-3300; Marshall (716) 235-7620;
Schweber (716) 424-2222;
Syracuse: Marshall (607) 798-1611.

NORTH CAROLINA: Arrow/Kierulff (919) 876-3132,
(919) 725-8711; Hall-Mark (919) 872-0712;
Marshall (919) 878-9882; Schweber (919) 876-0000.

OHIO: Cleveland: Arrow/Kierulff (216) 248-3990;
Hall-Mark (216) 349-4632; Marshall (216) 248-1788;
Schweber (216) 464-2970;
Columbus: Hall-Mark (614) 888-3313;
Dayton: Arrow/Kierulff (513) 435-5563;
Marshall (513) 898-4480; Schweber (513) 439-1800.

OKLAHOMA: Arrow/Kierulff (918) 252-7537;
Schweber (918) 622-8003.

OREGON: Arrow/Kierulff (503) 645-6456;
Marshall (503) 644-5050; Wyle (503) 640-6000.

PENNSYLVANIA: Arrow/Kierulff (412) 856-7000,
(215) 928-1800; GRS Electronics (215) 922-7037;
Marshall (412) 963-0441; Schweber (215) 441-0600,
(412) 963-6804.

TEXAS: Austin: Arrow/Kierulff (512) 835-4180;
Hall-Mark (512) 258-8848; Marshall (512) 837-1991;
Schweber (512) 339-0088; Wyle (512) 834-9957;
Dallas: Arrow/Kierulff (214) 380-6464;
Hall-Mark (214) 553-4300; Marshall (214) 233-5200;
Schweber (214) 661-5010; Wyle (214) 235-9953;
Zeus (214) 783-7010;
El Paso: Marshall (915) 593-0706;
Houston: Arrow/Kierulff (713) 530-4700;
Hall-Mark (713) 781-6100; Marshall (713) 895-9200;
Schweber (713) 784-3600; Wyle (713) 879-9953.

UTAH: Arrow/Kierulff (801) 973-6913;
Hall-Mark (801) 972-1008; Marshall (801) 485-1551;
Wyle (801) 974-9953.

WASHINGTON: Arrow/Kierulff (206) 575-4420;
Marshall (206) 486-5747; Wyle (206) 881-1150.

WISCONSIN: Arrow/Kierulff (414) 792-0150;
Hall-Mark (414) 797-7844; Marshall (414) 797-8400;
Schweber (414) 784-9020.

CANADA: Calgary: Future (403) 235-5325;
Edmonton: Future (403) 438-2858;
Montreal: Arrow Canada (514) 735-5511;
Future (514) 694-7710;
Ottawa: Arrow Canada (613) 226-6903;
Future (613) 820-8313;
Quebec City: Arrow Canada (418) 871-7500;
Toronto: Arrow Canada (416) 672-7769;
Future (416) 638-4771; Marshall (416) 674-2161;
Vancouver: Arrow Canada (604) 291-2986;
Future (604) 294-1166.

TEXAS INSTRUMENTS

Customer Response Center

TOLL FREE: (800) 232-3200

OUTSIDE USA: (214) 995-6611
(8:00 a.m. – 5:00 p.m. CST)

TEXAS INSTRUMENTS

Printed in U.S.A., March 1989
2534185-9704

SPNU008A