XDS512RL / Tek Emulation System User's Guide

Literature Number: SPRU259B
July 1998

IMPORTANT NOTICE

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

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale 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. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any 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 of 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 this interference.

Preface

Read This First

About This Manual

This manual explains how to use the XDS512RL/Tek emulation system, which is a tool for obtaining detailed signal trace data from your target system. The emulation system adds breakpoint, trace, and timing functionality to the Texas Instruments (TI™) C source debugger. These functions allow you to accomplish the following tasks:

- ❑ Monitor the CPU
- ❑ Benchmark performance
- ❑ Gather precise timing information
- ❑ Generate more compact and efficient code
- ❑ Improve debugging efficiency

Information About Cautions and Warnings

This manual contains cautions and warnings.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

Related Documentation From Texas Instruments

The following books contain information related to the XDS512 / Tek emulation system and the JTAG cable. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

XDS512RL User's Guide (literature number SPRU259) describes the components of the XDS512RL / Tek emulation system. This User's Guide documents how to use the XDS512RL emulator and TLA704 logic analyzer with the the BTT GUI and TMS320C27xx microprocessor support package to acquire trace data from your target system.

XDS512RL Getting Started Guide (literature number SPRU256) explains how to connect and install the hardware and software components of the XDS512RL / Tek emulation system, including the XDS512RL emulator, TLA704 logic analyzer, the BTT application, and the TMS320C27xx microprocessor support package.

JTAG/MPSD Emulation Technical Reference (literature number SPDU079) provides the design requirements of the XDS510™ emulator controller, discusses JTAG designs (based on the IEEE 1149.1 standard), and modular port scan device (MPSD) designs.

TMS320C27xx C Source Debugger User's Guide (literature number SPRU214) tells you how to invoke the TMS320C27xx emulator and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

T320C2700 Customizable Digital Signal Processor (cDSP™) Core (literature number SPRS057) data sheet contains the electrical and timing specifications for these devices.

TMS320C2700–E1 Digital Signal Processor In-Circuit Emulation Device (literature number SPRS062) data sheet contains the pinout, signal descriptions, as well as electrical and timing specifications for the device.

TMS320C27x XDS512RL Support Package User's Guide is provided by Tektronix, Inc. and documents the TMS320C27x microprocessor support package.

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/pic/home.htm
DSP Solutions	http://www.ti.com/dsps
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm
Microcontroller Home Page	http://www.ti.com/sc/micro
Networking Home Page	http://www.ti.com/sc/docs/network/nbuhomex.htm
MOS Memory Home Page	http://www.ti.com/sc/memory
Military Memory Products Home Page	http://www.ti.com/sc/docs/military/product/memory/mem_1.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
Microcontroller Hotline	(281) 274-2370	Fax: (281) 274-4203	Email: micro@ti.com
Microcontroller Modem BBS	(281) 274-3700 8-N-1		
DSP Hotline	(281) 274-2320	Fax: (281) 274-2324	Email: dsph@ti.com
DSP Modem BBS	(281) 274-2323		

DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs

Networking Hotline — Fax: (281) 274-4027
Email: TLANHOT@micro.ti.com

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32

Email: epic@ti.com

Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68
English	+33 1 30 70 11 65
Francais	+33 1 30 70 11 64
Italiano	+33 1 30 70 11 67

EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 4 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	

Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type “Go TIASP”	

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

Trademarks

320 Hotline On-Line, XDS510, and XDS512RL are trademark of Texas Instruments Incorporated.

HP-UX, HP 9000 Series 700, and PA-RISC are trademarks of Hewlett-Packard Company.

IBM and PC are registered trademarks of International Business Machines Corp.

Intel EtherExpress is a trademark of Intel Corp.

MICTOR is a trademark of AMP Incorporated.

PCAnywhere is a trademark of the Symantec Corp.

Tektronix and Tek are trademarks of Tektronix, Inc.

Windows is a registered trademark of Microsoft Corporation.

FCC 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 of 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 this interference.

Contents

Figures

Tables

Chapter 1

Overview of the XDS512RL / Tek Emulation System

The XDS512RL / Tek emulation system is a suite of debugging tools that adds breakpoint, trace, and timing (BTT) functionality to the development system that includes the XDS510 and XDS512RL. These functions allow you to:

- Monitor the CPU
- Monitor the program, read, and write buses and their respective bus qualifier signals, and to generate external signals to the Tektronix logic analyzer
- Benchmark performance
- Gather precise timing information
- Generate smaller, more efficient code
- Improve debugging efficiency

User-defined events control most activity in the XDS512RL / Tek emulation system. Events are defined by addresses or data patterns for any combination of buses, execution, and memory cycle types. When defined events occur, the emulation system can:

- Start, stop, or disable tracing
- Generate hardware breakpoints
- Generate external signals to the Tektronix logic analyzer
- General external signals to trace equipment such as a digital oscilloscope

This chapter describes the features and components of the XDS512RL / Tek emulation system and how the system operates in a debugging environment.

Topic **Page**

1.1 Features and Components of the XDS512RL / Tek Emulation System

The XDS512RL / Tek emulation system includes these features:

- ❑ 32K-byte real-time trace buffer
- ❑ Event recognition
- ❑ Trace acquisition and control
- ❑ Multiple hardware breakpoints
- ❑ Hardware and software timestamping
- ❑ External triggers for test equipment synchronization
- ❑ Ability to load/save trace configurations and samples

The XDS512RL / Tek emulation system monitors and records bus and cycle-type activity on the target device at the full operating speed of the target device.

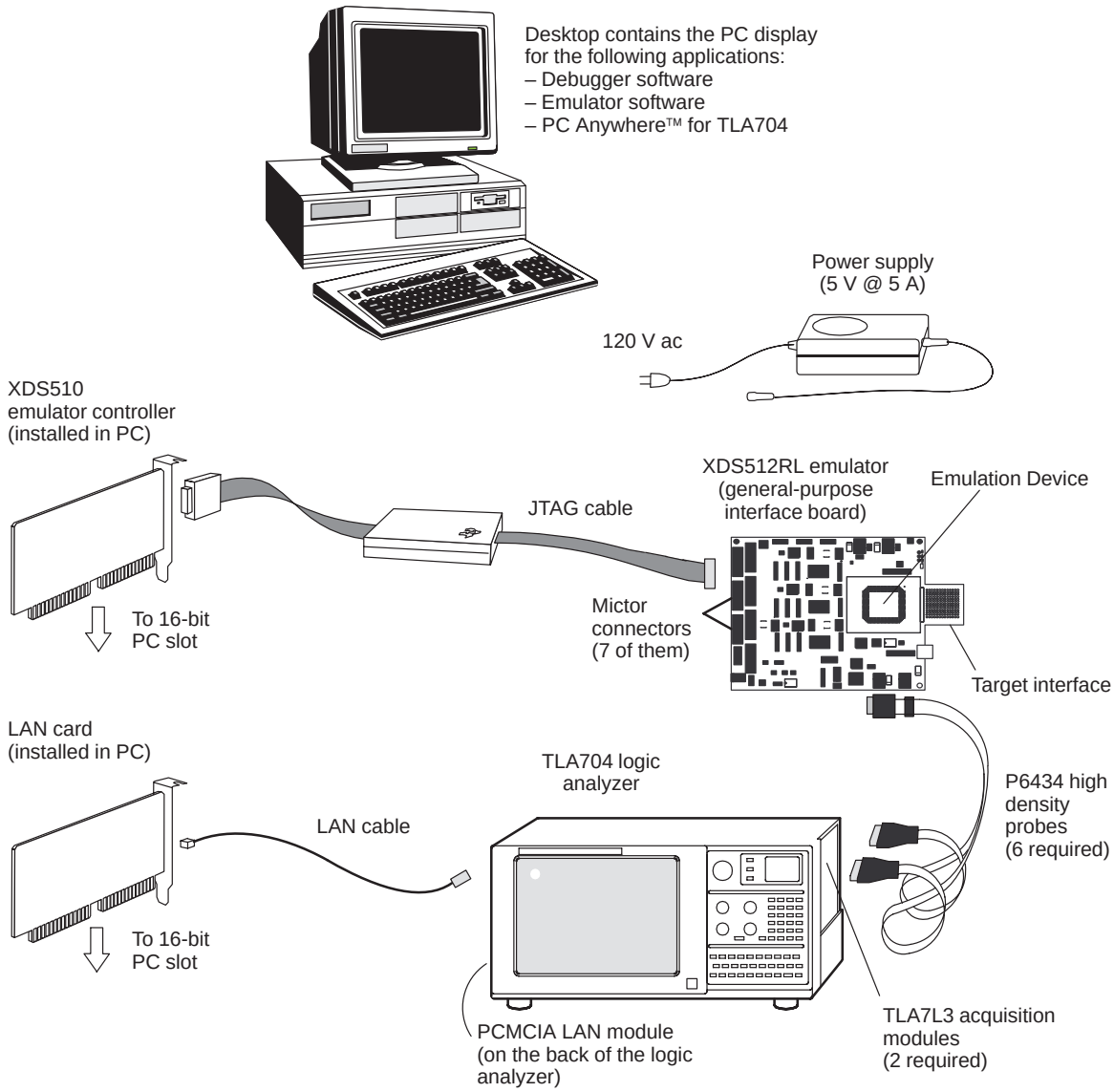
The TMS320c27x emulation device provides visibility into core operations through additional pins that monitor internal buses and cycle-type signals that are not available on production devices. These additional signals reveal internal state and bus information that improves debugging efficiency.

The XDS512RL / Tek emulation system consists of the following components:

- ❑ The PC™ display for the BTT software, C source debugger, and PCAnywhere™ for displaying the microprocessor support package that resides on the Tektronix logic analyzer model TLA704. This portion of the system must include a LAN card dedicated to networking the PC with the Tektronix logic analyzer.
- ❑ An XDS510 emulator controller board and a JTAG cable that connects the XDS510 to the XDS512RL emulator board
- ❑ The XDS512RL emulator board with a ’C27x emulation device
- ❑ A Tektronix logic analyzer with ’C27x microprocessor support package software, six P6434 high-density probes, two TLA7L3 acquisition modules configured for merged operation, and a PCMCIA LAN card dedicated to networking the TLA704 with the PC.
- ❑ A 5-V @ 5-A external power supply

Figure 1–1 shows how the components of the XDS512RL / Tek emulation system connect.

Figure 1–1. XDS512RL/Tek Emulation System



1.2 Understanding the Debugging Environment

The XDS512RL / Tek emulation system supports code-development and debugging activity for the ’C27x DSP devices. The XDS512RL emulator board supports debugging for stand-alone mode and slave mode operation. In stand-alone operation, your target is operating in the socket on the XDS512RL emulator board. Slave operation is when the XDS512RL emulator board is interfaced to an actual target system.

The typical debugging environment for a ’C27x includes a PC C source debugger, BTT application, and an XDS510 emulator controller that connects to a ’C27x device through an XDS512RL emulator. In turn, the XDS512RL emulator is connected to the Tektronix logic analyzer by six P6434 high-density probes.

When the C source debugger is set up to use BTT functionality, debugger actions cause the XDS512RL / Tek emulation system to respond to the activity on the ’C27x emulation device. For example, if a software breakpoint occurs on the target, the TLA704 is updated with the new trace information.

Chapter 2

How the Emulation System Works

This chapter explains how the components of the XDS512RL emulation system work together to acquire trace data from a TMS320C27x device.

Topic **Page**

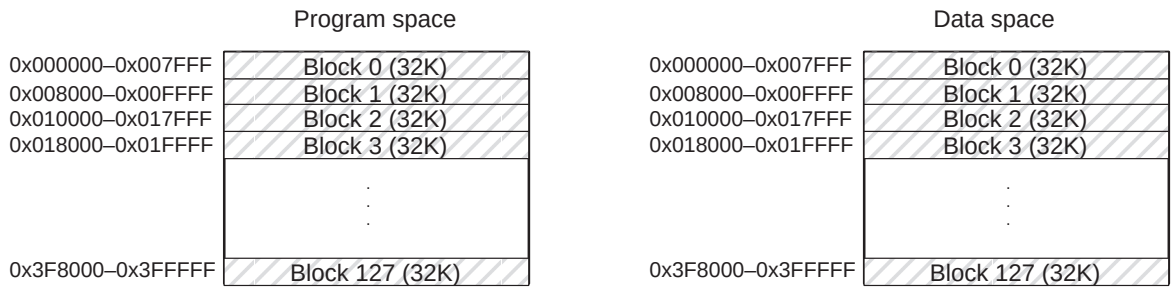
2.1 Range Logic Overview

The range logic on the XDS512RL emulator board lets you monitor the program, read, and write address buses on your target system. You can program the range logic with data, defined as events, that the range logic compares to the data on the target address buses. When the range logic detects a match between the data on the target system's address buses and the data you have programmed into it, it can trigger external trace equipment, such as the TLA704 logic analyzer or a digital oscilloscope.

'C27x target memory contains 128 32K blocks in program space and 128 32K blocks in data space (see Figure 2–1). The range logic allows you to monitor 24 of these blocks (768K) at any given time. The 24 range logic blocks are subdivided as follows:

- Eight 32K blocks to monitor program activity
- Eight 32K blocks to monitor data-read activity
- Eight 32K blocks to monitor data-write activity

Figure 2–1. 32K Blocks in TMS320C27x Memory



2.2 Range Logic Description

The range logic works differently for program space than it does for read and write space. The rest of this section explains how the range logic works for program, read, and write space in ungrouped mode.

2.2.1 The Program Space Range Logic

The program space range logic bank is divided into eight 32K blocks (0–7). You can have each block monitor any 32K range of addresses within the TMS320C27x program space. You use these program space range logic blocks to detect events on the program address bus and to perform actions upon detecting those events. Upon detecting the events, the program space range logic asserts one or more of the following output signals:

Program signal	XDS512RL probe point
ProgramLow	(output to TLA704)
Program4	JP4–1
Program5	JP4–3
Program6	JP4–4
Program7	JP4–5

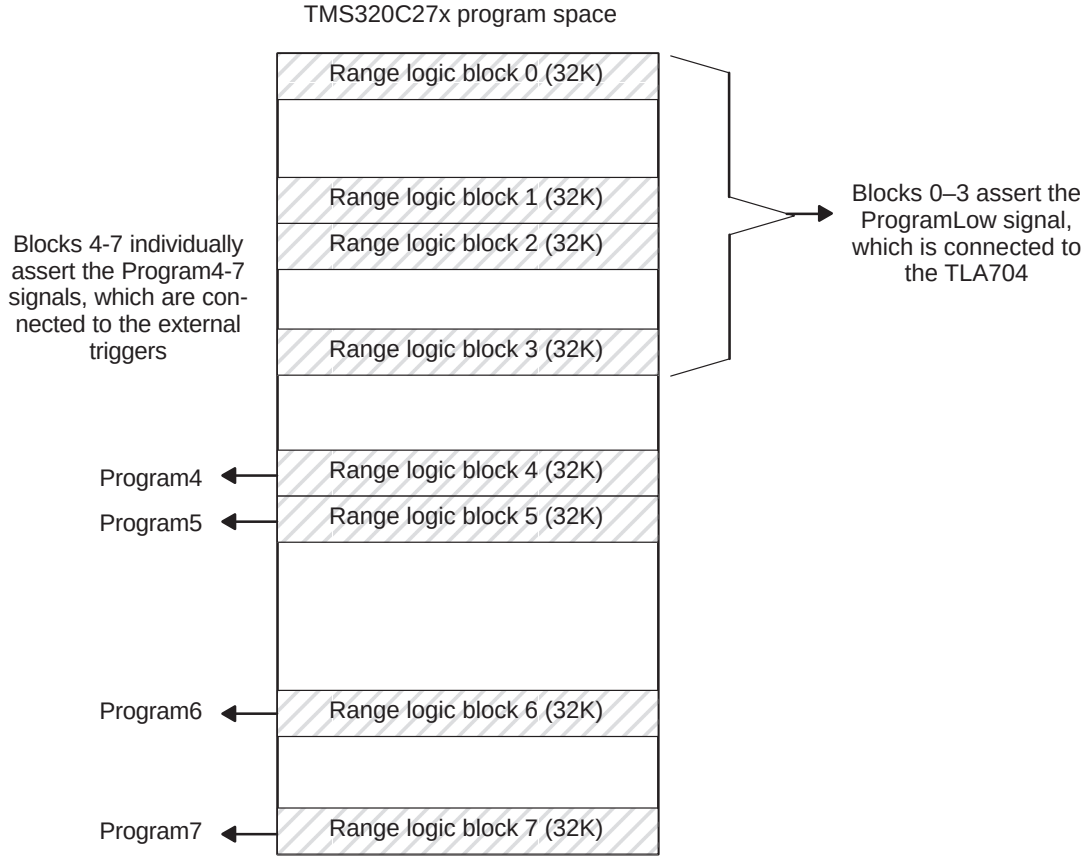
The ProgramLow signal is output to the TLA704 and the Program4-7 signals are output to external test points on the XDS512RL board. You can use Program4–7 for triggering external test equipment, or for other special uses.

Each range logic block is tied to a specific signal or signals. The ProgramLow signal is assigned to four of the eight blocks (0–3). You can use these blocks of range logic to output the ProgramLow signal to the TLA704. When the data on your target's program address bus matches any of the defined events in blocks 0–3, the range logic outputs the ProgramLow signal to the TLA704. In turn, you can configure the TLA704 to acquire trace data when it detects a ProgramLow signal from the range logic.

The Program4-7 signals are assigned to program space range logic blocks 4-7. Each block can assert the respective signal; for example, block 4 can assert Program4. When the data on your target's program address bus matches any of your defined events in blocks 4–7, the range logic outputs the corresponding Program4-7 signal to the external equipment attached to the test points. Figure 2–2 shows an example of range logic blocks that have been tied to 32K address ranges in program space. The figure also indicates the range logic output signal or signals that are associated with each of the range logic

blocks. This signal assignment is for ungrouped mode. For a slightly different signal assignment, see section 2.3, *Grouping Range Logic Output Signals*, on page 2-7.

Figure 2–2. Program Space Range Logic Functionality In Ungrouped Mode



2.2.2 Read and Write Space Range Logic

The range logic banks for data reads and data writes function identically and are described here together. The read space bank and write space bank are divided into eight 32K blocks each (0–7). You can have each block monitor any 32K range of read/write addresses within the TMS320C27x read/write space address ranges. You use the read and write space range logic blocks to detect events on the read/write address buses, and to perform actions upon detecting those events. Upon detecting the events on the target, the range logic asserts one or more of the following output signals:

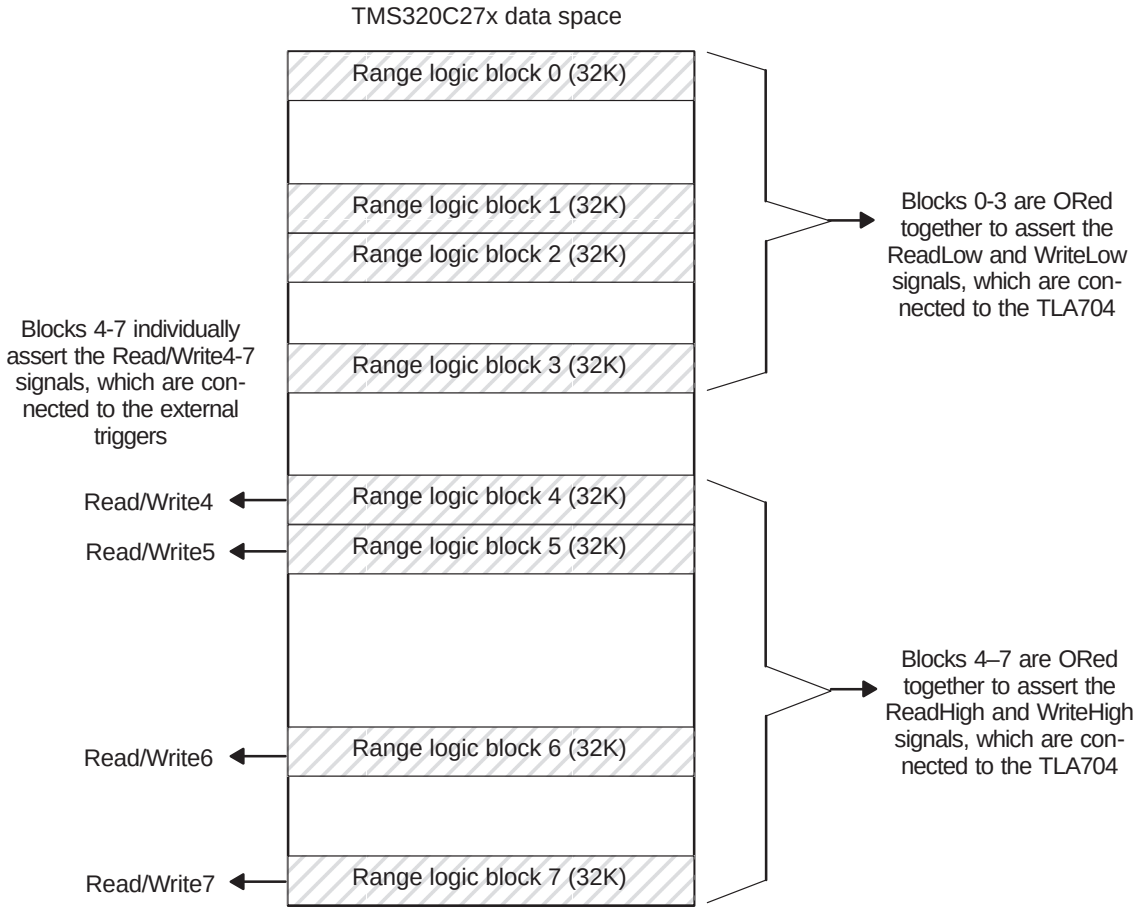
Read signal	XDS512RL probe point	Write signal	XDS512RL probe point
ReadLow	(output to TLA704)	WriteLow	(output to TLA704)
ReadHigh	(output to TLA704)	WriteHigh	(output to TLA704)
Read4	JP5–1	Write4	JP6–1
Read5	JP5–3	Write5	JP6–3
Read6	JP5–4	Write6	JP6–4
Read7	JP5–5	Write7	JP6–5

The Read/WriteLow and Read/WriteHigh signals are output to the TLA704 and the Read/Write4-7 signals are output to external test points on the XDS512RL board. You can use Read/Write4–7 for triggering external test equipment, or for other special uses.

Each block of read/write space range logic is tied to a specific signal or signals. The Read/WriteLow signal is assigned to four of the eight blocks (0–3), and the Read/WriteHigh signal is assigned to the other four blocks (4–7). When the data on your target's read/write address bus matches any of the defined events in blocks 0–3, the range logic sends the Read/WriteLow signal to the TLA704. When the data on the read/write address bus matches any of the defined events in blocks 4–7, the range logic sends the Read/WriteHigh signal to the TLA704. You can configure the TLA704 to acquire trace data when it detects a Read/WriteLow or Read/WriteHigh signal from the range logic.

The Read/Write4-7 signals are tied to read/write blocks 4–7 and are sent to external test points on the XDS512RL board. Whenever an event triggers any one of these signals, the Read/WriteHigh signal is also triggered. Figure 2–3 shows an example of range logic blocks that have been tied to 32K address ranges in data space and shows the output signals associated with each range logic block. This signal assignment is for ungrouped mode. For a slightly different signal assignment, see section 2.3, *Grouping Range Logic Output Signals*, on page 2-7.

Figure 2–3. Data-Space Read/Write Range Logic Functionality In Ungrouped Mode



2.3 Grouping Range Logic Output Signals

In grouped mode, the range logic outputs the Program/Read/WriteLow and Read/WriteHigh signals when any event is true. Figure 2–4 and Figure 2–5 show conceptual diagrams of grouped functionality and ungrouped functionality, respectively. Figure 2–6 shows an example of range logic blocks that have been tied to 32K address ranges and indicates the range logic output signal or signals that are associated with each of the range logic blocks.

Figure 2–4. Conceptual Diagram of Grouped Range Logic Functionality

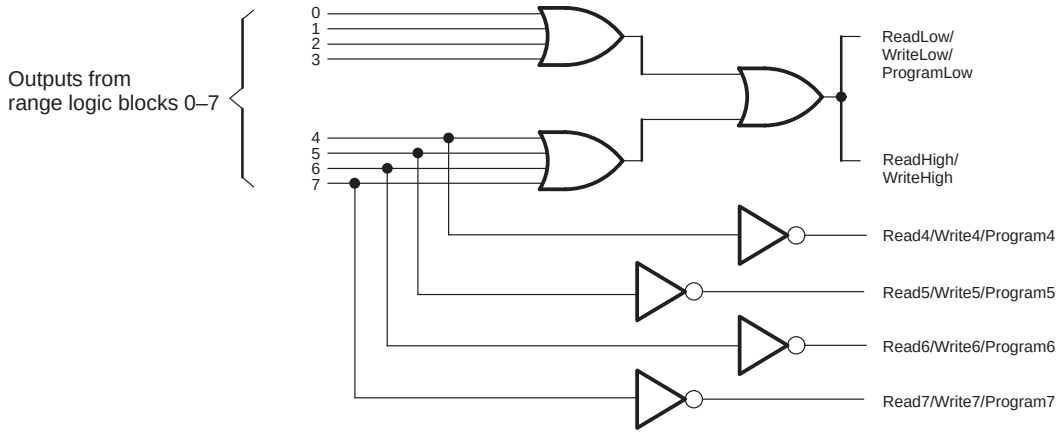


Figure 2–5. Conceptual Diagram of Ungrouped Range Logic Functionality

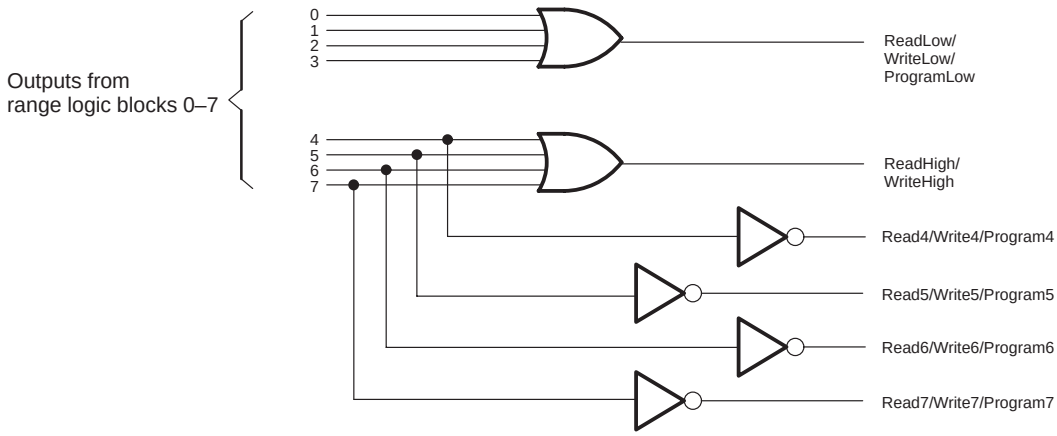
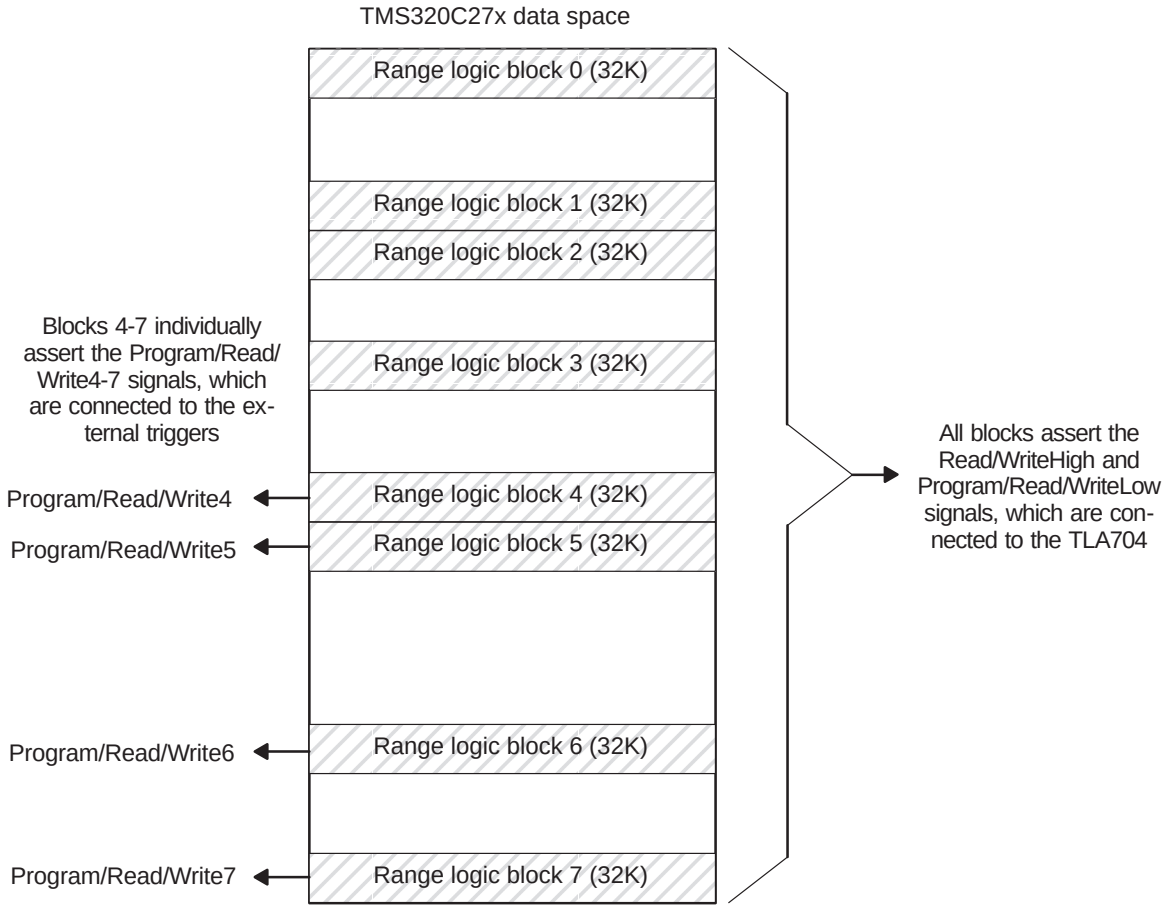


Figure 2–6. Read/Write Space Range Logic Functionality in Grouped Mode



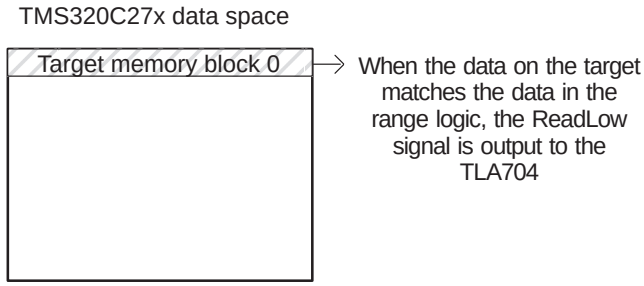
2.4 Using the Range Logic

This section contains a high-level example of how you use range logic blocks to acquire data for a specific event occurring on the program/read/write address buses of the TMS320C27x device.

Suppose you want to have the TLA704 acquire trace data that shows what is happening on the target during a read access within the first 32K of data space. First, you would use the BTT application to program the range logic to output the ReadLow signal to the TLA704 upon detecting a read access within the first 32K of TMS320C27x read address space (see). Next, you would set up the TLA704 to acquire and store data when it receives the ReadLow signal from the range logic. Last, you would use the TMS320C27x source code debugger to run the code on the target system.

When data on the target's read address bus matches the event you programmed into the range logic, the range logic outputs the ReadLow signal to the TLA704. The TLA704 then acquires the data from the target system and stores it in a trace buffer. You can then use the TLA704 to display the trace data for you to examine.

Example 2–1. ReadLow Asserted By a Read Within the First 32K of Data Space



If you want the read access to trigger both the ReadLow and ReadHigh signals, use the grouping feature of the BTT application. For more information about grouping, see section 2.3, *Grouping Range Logic Output*, on page 2-7.

For more information about the BTT application, see Chapter 4, *The BTT Application*.

2.5 The BTT Application

The BTT application is the interface you use to define the conditions you want to trace. You program the conditions into the range logic as a trace language program. This program requires you to define terms and events that define the conditions you want to trace. This section explains the following tasks:

- ❑ Defining Terms and Events for Trace Language Programs.
- ❑ Building your Trace Language program
- ❑ Programming the XDS512 range logic

2.5.1 Defining Terms and Events for a Trace Language Program

A trace language program contains the trace conditions you want the range logic to detect. Trace language programs are made up of terms and events. A term compares an address bus's current value to one or more addresses or address ranges that you specify and/or checks for a CPU condition or conditions that you specify. Events are ORed terms.

Defining Terms

You use terms to monitor specific address buses or CPU conditions. The BTT application has a term editor that lets you create and edit terms. After you create a term, it is displayed by the BTT application so that you always have a list of your current terms. See the online help for the procedure for defining a term.

Defining events

An event is a group of ORed terms that tells the range logic when to output a specific signal. An event is true when the data on the target matches any one of the term definitions within the event. When an event is true, the range logic outputs the event's assigned signal to the external trace equipment. The BTT application has an event editor that lets you create and edit event definitions. After you create an event, you can assign any of your terms to the event. See the online help for the procedure for defining an event.

2.5.2 Building Trace Language Programs

After you have created your terms and defined your events, you can build a trace language program. You build a Trace Language program by pressing  on the BTT application toolbar or by selecting Build from the Program menu.

When you use the build command, the BTT application verifies that there are no errors or conflicts in your event and term definitions. If you have errors in your event or term definitions, the BTT application displays a list of errors. You can then double-click on an individual error to display and correct the erroneous definition.

If your event and term definitions are valid, the BTT application compiles them into a trace language program that you can program into the range logic on the XDS512RL emulator board. You can save your trace language programs in a project directory of your choice. Trace language programs are saved with a .btt extension.

2.5.3 Programming the Range Logic

When you want to program the range logic with a trace language program, press  on the toolbar or select Program from the Program menu. The program command programs the range logic with the current trace language program.

When you use the program command, the BTT application builds and verifies the trace language program before programming it into the range logic. If the BTT application detects errors in the trace language program or if it cannot initialize the XDS512RL, then the program button and menu selection are grayed out.

If the BTT application cannot initialize the XDS512RL, it displays an error box with a descriptive message that explains the initialization problem.

For more information about the BTT application, see Chapter 4, *The BTT Application.*

2.6 The TLA704 and the TMS320C27x Microprocessor Support Package

The Tektronix logic analyzer model TLA704 is the instrument that acquires trace data from the TMS320C27x device on the XDS512RL emulator board. The logic analyzer software must be configured specifically to acquire data from the XDS512RL emulator board. This configuration includes defining the channel groups that the P6434 probes use when transmitting the data and setting timing algorithms for the TLA704's acquisition rate. The 'C27x microprocessor support package configures all of these settings for you so that you can focus on defining your acquisition criteria.

2.6.1 The TMS320C27x Microprocessor Support Package

The 'C27x microprocessor support package is software installed on the TLA704 that performs the following tasks:

- Configures the TLA704 acquisition modules by naming channels and groups for acquisitions from the XDS512RL and by assigning channels to groups
- Installs a clocking algorithm to sample valid data from the XDS512RL trace interface (only flattened mode is supported)
- Installs symbol tables for cycle type and size channel groups. You can use these symbols to define trigger algorithms and show mnemonics in the listing displays for analyzing trace data.
- Installs the 'C27x mnemonic disassembler for analyzing trace data (only flattened mode is supported)
- Provides a translation utility for extracting source code symbols from the COFF object load modules. Once extracted, the symbols are put into TLA704 symbol tables for analysis of trace data.

For more information about the TMS320C27x microprocessor support package, see the *TMS320C27x XDS512RL Support Package User's Guide* provided by Tektronix, Inc.

2.6.2 Channel Groups

The instrument setup file in the ’C27x support package configures the TLA704 to acquire trace data from the XDS512RL. This configuration forms channel groups and assigns named signals to the groups. The configuration also assigns a radix for each group and, where appropriate, provides a symbol table. The symbol tables are loaded automatically into the instrument setup when the ’C27x support package is loaded, or when an instrument setup is loaded that uses this same support package. A summary of the groups and their corresponding symbol tables provided by this instrument setup are available in the *TMS320C27x XDS512RL Support Package User’s Guide* provided by Tektronix, Inc.

2.6.3 Translating Symbol Tables

For the TLA704 to recognize and display trace data for the source code executing on your target system, it must be able to recognize the symbols in the source code. The ’C27x microprocessor support package installs a symbol translation utility, teksym.exe, that translates the symbols from your COFF files into a format that the TLA704 can recognize. When you run your COFF file through the symbol translation utility, the utility creates four output files with the same name as your COFF file but with the extensions described in the following table. After you translate your symbol tables, you can access the symbol files from the TLA704 by browsing to your project directory on your PC.

Extension	File Type	Description
.pps	program pattern symbols	This type of file contains...
.prs	program range symbols	This type of file contains...
.dps	data pattern symbols	This type of file contains...
.drs	data range symbols	This type of file contains...

2.6.4 Using the TLA704 to Acquire Data

The TLA704 acquires data from the target system when it is triggered by a signal or signal pattern from the range logic. The software on the TLA704 lets you define conditions that specify when you want the TLA704 to acquire data and save it as a trace file. These conditions look for a specific signal output, or pattern of signal outputs, from the range logic on the XDS512RL.

An acquisition session with the TLA704 consists of the following tasks:

- ❑ Loading an instrument setup file
- ❑ Creating an acquisition algorithm
- ❑ Acquiring data
- ❑ Saving your setup file

Loading an instrument setup file

An instrument setup file configures the TLA704 to acquire data from the XDS512RL and allows you to view and manipulate your trace data. The microprocessor support package comes with a default instrument setup file, C27xsetup.tla, that you can use as a template for configuring your unique trace activities. You can create a separate instrument setup file for each trace case. Once you create an instrument setup file for your trace scenario, you can save it with a unique name to help you identify it later. After you load your instrument setup file, you define the TLA704's trigger conditions—these conditions tell the TLA704 when to acquire data. For more information about instrument setup files, see the *TMS320C27x XDS512RL Support Package User's Guide* provided by Tektronix, Inc. For information about defining trigger conditions on the TLA704, see the TLA704's online help and your Tektronix documentation.

Setting Triggers

The TLA704 has acquisition software that lets you create a trigger program. A trigger program contains a series of events and actions that define when to acquire and store data. The conditions configure the TLA704 to acquire and store data when it detects specific signals or patterns of signals from the range logic. The conditions that you define work similarly to the term and event definitions of the BTT application.

The TLA704's online help provides an example to guide you through the basic steps of using the acquisition software. For more information about using the acquisition software, see your Tektronix documentation.

Acquiring Data

Once you have set up your trace conditions on the BTT application and the TLA704, it is time to run your source code and acquire your trace data.

At this point in the procedure, you use the 'C27x source code debugger to load your application code into the target and run the code on the target. When you run your source code on the 'C27x, the range logic looks for matches between the data in your trace language program and the data on the target. When the range logic detects a match, it outputs the appropriate signal; which triggers the TLA704 to begin acquiring data and storing it in your setup file.

Chapter 3

Tutorial for the XDS512RL Emulation System

This tutorial guides you through the process of acquiring trace data from your target system. This chapter assumes that you know how to use the 'C27x code generation tools and debugger.

To use this tutorial, you must have correctly installed the following software:

- ❑ 'C27x code generation tools
- ❑ 'C27x emulator/debugger
- ❑ BTT application
- ❑ 'C27x microprocessor support package

Topic **Page**

3.1 How to Use This Tutorial

This tutorial guides you through the trace process. The goal of this tutorial is to familiarize you with the process, not to provide an in-depth explanation about the hardware and the software involved. For more detailed information, see the rest of this manual.

Some of the procedures in this tutorial would normally require that you write some application code. To expedite the tutorial, we will use some provided sample code. To access the sample code, you must have already installed the TMS320C27x tools. If you have not installed the tools, please do so before proceeding with this tutorial.

The procedures necessary for tracing your target system are explained in the following order:

- **Configuring the tools**
  - Initializing the XDS512RL with the BTT application
  - Restoring the TLA704 to its default instrument setup
  - Restoring the ’C27x support package to its default setup
- **Test and debug preparation**
  - Creating application software
  - Translating symbol tables
  - Loading your COFF file into the target system
  - Assigning symbolic radix and setting the display mode
- **Developing trace capture algorithms and acquiring data**
  - Creating a trace language program with the BTT application
  - Building a trace language program with the BTT application
  - Programming the range logic with the BTT application
  - Creating a trace capture algorithm for the TLA704
  - Using the TLA704 to execute a trace capture algorithm
  - Viewing the disassembly of trace capture data in the Setup window
  - Saving your trace capture algorithm and data

3.2 Configuring the Tools

Before you can begin acquiring trace data, you must set up your tools. This section walks you through the steps necessary to configure the tools.

3.2.1 Initializing the XDS512RL With the BTT Application

For a successful acquisition session, the BTT application must be able to communicate with the XDS512RL. Upon invocation, the BTT application initializes the XDS512RL. Make sure the XDS512RL is powered on before you invoke the BTT application. If the BTT application cannot initialize the XDS512RL, it displays an error message that tells you what the problem is.

3.2.2 Restoring the TLA704 to its Default Instrument Setup

To restore the TLA704 to its default instrument setup, follow these steps:

1) Turn on your TLA704 if it is not already on.

2) From File menu on the TLA704's main window, select Default System.

3) Click OK to restore the default instrument setup

3.2.3 Loading the TMS320C27x Support Package Default Setup

To restore the C27x micro support package to its default setup, follow these steps:

1) From the *File* menu on the TLA704's main window, select Load System.

2) Browse to C:\ProgramFile\TLA700\Supports\C27x and select C27xsetup.

3) Click Open. After the setup file is loaded, your TLA704's display should look similar to Figure 3–1.

Figure 3–1. TLA704 Display After Loading The Support Package

3.3 Test and Debug Preparation

Your next step in the trace process involves the ’C27x code generation tools. Creating your application code and preparing for testing and debugging consists of the following tasks:

- ❑ Creating application code
- ❑ Translating the symbol tables for the TLA704
- ❑ Loading your COFF file into the target system
- ❑ Assigning a symbolic radix

For more information about the ’C27x code generation tools, see your TMS320C27x Compiler, Assembler, and Debugger User’s guides.

3.3.1 Creating Application Code

This is the point in the trace process where you code your assembly or C application using the C27x Code Generation Tools. To expedite this tutorial, we are going to use the sample file, sample.out, that is provided with the TMS320C27x Debugger.

3.3.2 Translating Symbol Tables

To allow the TLA704 to use the symbols in your code, you must translate your COFF file’s symbol tables. Once translated and made accessible to the TLA704, you can use these symbols in trace data display and trace capture algorithms. To translate your COFF file’s symbol tables, you must have included the teksym utility, *teksym.exe*, into your COFF build script. To expedite this tutorial, the following symbol table files are provided with the ’C27x microprocessor support package:

- ❑ sample.prs
- ❑ sample.pps
- ❑ sample.drs
- ❑ sample.dps

For more information about the Teksym Utility, see the *TMS320C27x XDS512RL Support Package User’s Guide* provided by Tektronix, Inc.

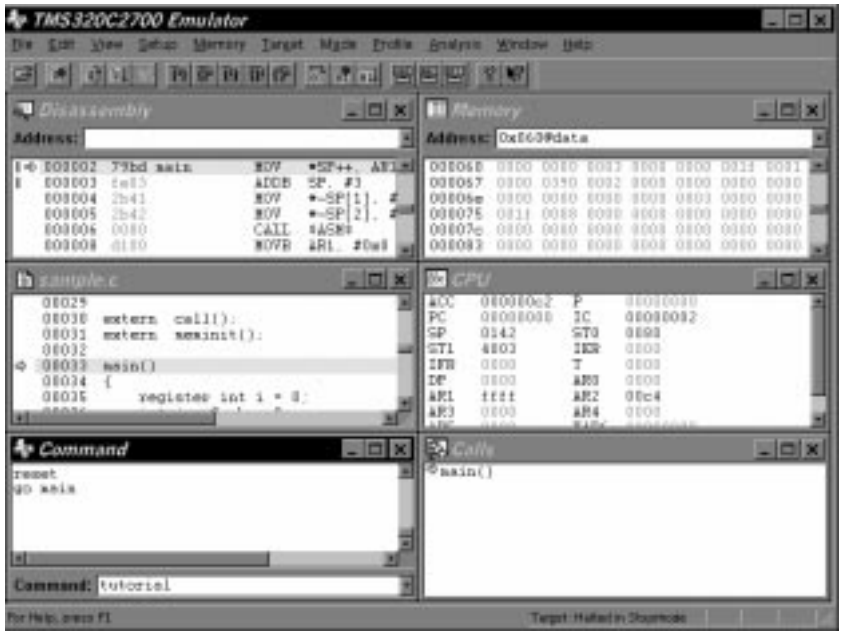
3.3.3 Loading your COFF file into the target system

If your TMS320C27x debugger is not running, you will need to invoke it for this part of the tutorial. once you have invoked your debugger and powered on your XDS512RL emulator board, type the following command in the debugger command-line:

```
tutorial
```

This command loads the tutorial COFF file, sample.out, from the tutorial directory into your target system. After you type this command, your debugger display should look similar to Figure 3–2.

Figure 3–2. The Debugger Display After Loading Sample.out



3.3.4 Assigning Symbolic Radix

The symbol table files you create with your build script provide the 'C27x support package with enough information to recognize the symbols from your COFF file; however, for the 'C27x support package to display your trace data symbol names, you must assign a symbolic radix to the groups you want to display. To assign a symbolic radix for the data groups you want to view in the 'C27x support package, follow this procedure:

1) Open the the TLA704's Listing window.

2) Right-click in the Listing window and select *Properties* from the context menu to display the Properties window.

3) Select the *Column* tab in the properties window.

4) From the pulldown menu in the Column field, select the *C27x PAddr* group. This is the group in the Listing window that displays the program address bus data.

5) From the pulldown menu in the Radix box, select *Symbolic*. This displays the information in Listing window's *C27x PAddr* column with a symbolic radix.

6) Click the *Symbol Table* button to display the the Select Symbol File dialog box.

7) In the *Look In* field, browse to the following path on your host PC where *C:*\ is the drive that contains your 'C27x tools:
C:\C27xDSP\EMU\TUTORIAL

8) From the File Types pulldown menu, select All Files.

9) Select *Sample.prs*. This is the symbol table file for your program address bus symbols.

10) Click Open to load the file. Now your Properties window should look similar to Figure 3–3 on the following page. Next, you will repeat this process for the *C27x RAddr* group.

11) From the Column field in the Properties window, select the *C27x RAddr* group. This is the group in the Listing window that displays the read address bus data.

12) Repeat steps 4–7.

13) Select *Sample.drs*. This is the symbol table file for your read address bus symbols.

14) Click Open to load the file. Next, you will repeat this process for the *C27x WAddr* group.

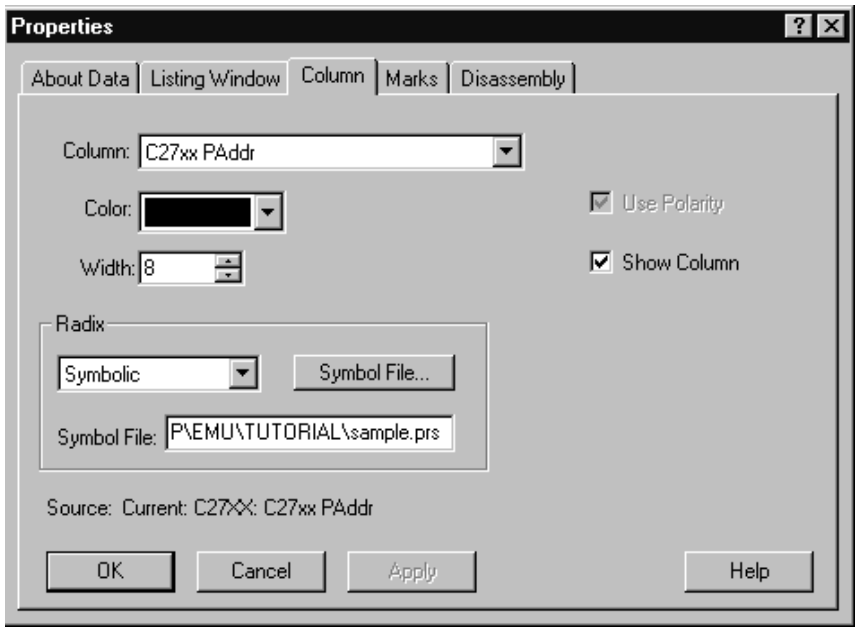
15) From the Column field in the Properties window, select the *C27x WAddr* group. This is the group in the Listing window that displays the write address bus data.

16) Repeat steps 4–7.

17) Select *Sample.drs*. This is the symbol table file for your read address bus symbols.

18) Click open to load the file

Figure 3–3. The Listing Window's Properties Dialog Box



3.3.5 Setting the Display Mode

Selecting symbolic radix and symbol tables displays your code's symbols in the address groups within the TLA704 trace display. To display these same symbols in the mnemonics group within the TLA704 trace display, you must select a display mode. To select a display mode, follow these steps:

1) Open the the TLA704's Listing window.

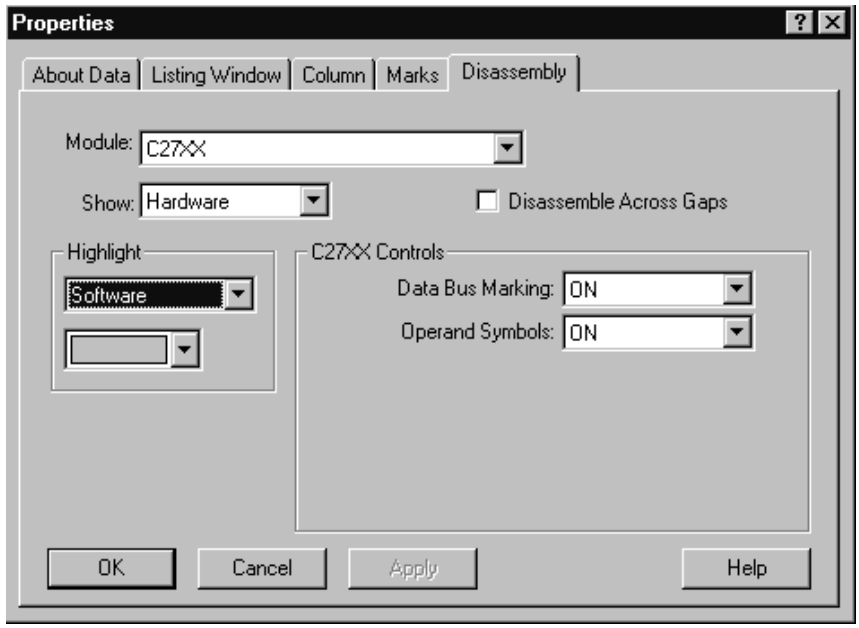
2) Right-click in the Listing window and select Properties from the context menu to display the Properties window.

3) Select the *Disassembly* tab in the properties window.

4) Within the 'C27x Controls box, select ON from the Operand Symbols pull-down menu. After this step, the properties window should look similar to Figure 3–4.

5) Click on OK to apply the settings and close the window.

Figure 3–4. The Disassembly Properties Window



3.4 Developing Trace Algorithms With the BTT Application and the TLA704

Now it's time to define your trace capture algorithms. Trace capture algorithms define the conditions you want to trace. You define trace capture algorithms with the BTT application and the software on the TLA704. This section walks you through the following procedures:

- ❏ Creating a trace language program with the BTT application
- ❏ Building a trace language program with the BTT application
- ❏ Programming the range logic
- ❏ Creating a trace capture algorithm for the TLA704
- ❏ Executing a trace capture algorithm on the TLA704

3.4.1 Creating a Trace Language Program With the BTT Application

Trace language programs contain your trace conditions and tell the range logic on the XDS512RL what data to look for on the TMS320C27x. Trace language programs consist of event and term definitions that specify your trace criteria. A term compares an address bus's current value to one or more addresses or address ranges that you specify and/or checks for a CPU condition or conditions that you specify. Events are ORed terms. Creating a trace language program consists of the following procedures:

- ❏ Creating an event
- ❏ Creating terms
- ❏ Defining an event
- ❏ Building a trace language program
- ❏ Programming the range logic

Creating Events

Let's say you want to store trace information for each of the following conditions:

- ❏ The program address bus accesses the *main* function
- ❏ The write address bus accesses the *aai* array

Your first step is to create events to which you can store these conditions. Follow these steps to create an event for the program bus:

1) From the toolbar, click  to display the Event Editor.

2) Click in the Event Name field and type **DetectMain**.

3) Right-click in the Range field and select **ProgramLow**.

4) Click in the Comments field and type **Detect main function**. After these steps, the Event Editor looks similar to Figure 3–5.

5) Click the OK button to save the event definition.

After you click OK, the event name *DetectMain* appears in the Events window of the BTT Application.

Figure 3–5. The Event Editor

Next, follow these steps to create an event for the write bus:

1) From the toolbar, click  to display the Event Editor.

2) Click in the Event Name field and type **WriteAAI**.

3) Right-click in the Range field and select **WriteLow**.

4) Click in the Comments field and type **Write accesses to aii array**. After this step, the event editor should look similar to Figure 3–6.

5) Click the *OK* button to save the event definition.

After you click OK, the event name *WriteAAI* appears in the Events window of the BTT Application.

Figure 3–6. The Event Editor After Defining WriteAAI

Creating Terms

Now that you have created your events, it's time to create the terms that define the events.

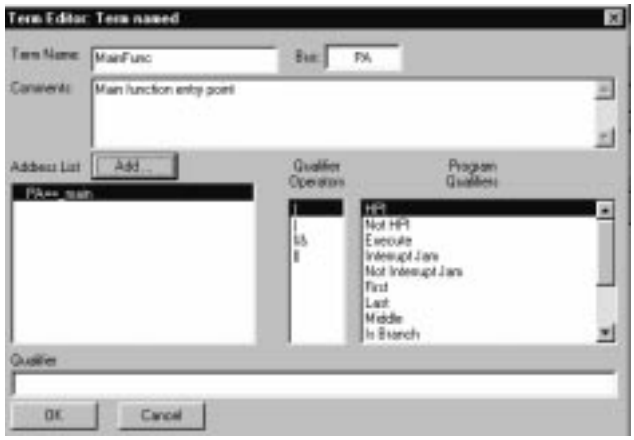
For our program bus event to serve it's purpose, we need to fill it with a term that detects when the address of the *main* function is on the program address bus. To create a term definition for this purpose, follow these steps:

1) From the toolbar, select  to display the Term Editor.
2) Click in the Term Name field and type **MainFunc**.
3) Right-click in the Bus field and select **PA**.
4) In the comments field, enter **Main function entry point**.
5) Click Add... to display the Address Editor.
6) Click the Address radio button to indicate that you are going to specify a single address, not a range of addresses.
7) Click Load Object File to display the Object File Selection dialog box.
8) Navigate to C:\C27xDSP\EMU\TUTORIAL and select *sample.out*.
9) Click Open. This closes the Object File Selection dialog box and populates the Program window of the Address Editor with the program symbols from *sample.out*.
10) Drag the _main symbol from the Program window to the Symbol field. At this point, the Address Editor looks similar to Figure 3–7.
11) Click OK. The Address Editor closes, and the new term definition, PA==_main, appears in the Address List of the Term Editor. The Term Editor now looks similar to Figure 3–8.
12) Click OK to save the term definition.

After you click OK, the new term appears in the Terms window of the BTT Application.

Figure 3–7. Address Editor After Defining Address for MainFunc

Figure 3–8. Term Editor After Creating MainFunc Term



Next, follow these steps to create a term that detects when the aai array is accessed by way of the write address bus:

13) From the toolbar, select  to display the Term Editor.

14) Click in the Term Name field and type **WRaai**.

15) Right-click in the Bus field and select **WA**.

16) In the Comments field, enter **Write access to aai array**.

17) Click Add... to display the Address Editor. The Data window displays the data symbols of *sample.out* (the last object file that was loaded).

18) Click the Range radio button to indicate that you are going to specify a range of addresses, not a single address. The Address Editor now provides fields for a Start Address and for an End Address or Length.

19) Define the Start Address by dragging the _aai symbol from the Data window to the Symbol field.

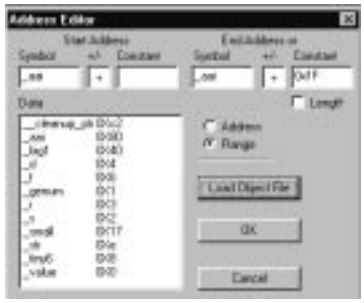
20) Define an End Address (_aai + 0x1F) by doing the following: First, drag the _aai symbol from the Data window to the Symbol field. Second, if necessary, right-click on the +/– field and choose Plus. Last, enter 0x1F in the Constant field. At this point, the Address Editor looks similar to Figure 3–9.

21) Click OK. The Address Editor closes, and the new term definition, WA[](_aai,_aai+0x1F), appears in the Address List of the Term Editor.

22) Click OK to save the term definition.

After you click OK, the new term appears in the Terms window of the BTT Application.

Figure 3–9. Address Editor After Defining Address for WRaai



Assigning Terms to an Event

Now that you have created your events and terms, it is time to assign the terms to the events. To assign terms to the events, follow these steps:

1) Select the *DetectMain* event in the Events window.

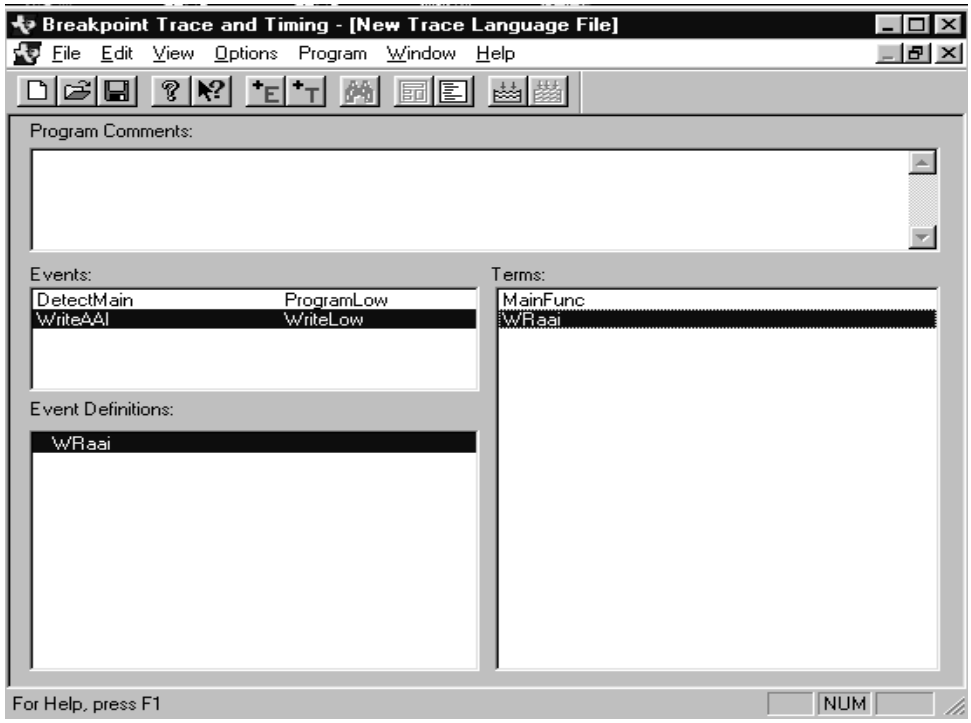
2) Assign the *MainFunc* term to the *DetectMain* event as follows: Select the *MainFunc* term and drag it to the Event Definitions window.

3) Select the *WriteAAI* event in the Events window.

4) Assign the *WRaai* term to the *WriteAAI* event as follows: Select the *WRaai* term and drag it to the Event Definitions window.

Now that you have defined the event, your BTT Application should look similar to Figure 3–10.

Figure 3–10. BTT Window With Defined Events



Building a Trace Language Program

After you have defined your terms and assigned them to events, you must build a them into a Trace Language program. To build your Trace Language program, perform this procedure:

1) From the toolbar, press  or select Build from the Program menu.

Programming the Range Logic

After you build your trace language program, you program it into the range logic. To program your Trace Language file into the range logic, perform this procedure:

1) From the toolbar, press  or select Program from the Program menu.

3.4.2 Creating a Trace Capture Algorithm for the TLA704

The TLA704 is the instrument that acquires the trace data. You must create a trace capture algorithm that tells the TLA704 which data to acquire and store. To create a trace capture algorithm for the TLA704, follow these steps:

Configuring Storage Definition

To create a trace capture algorithm, follow these steps:

1) From the System window, press  to display the Trigger window.

2) In the Storage field, select *Conditional* from the pulldown menu. This configures the TLA704 for conditional storage.

3) From the Storage box, click  to display the Storage Definition dialog box.

4) From the first pulldown menu, select Channel.

5) From the Second pulldown menu, select WriteLow.

6) Click the *Add* button at the bottom of the Storage Definition window.

7) Make sure the button to the left of the second storage definition group says *AND*.

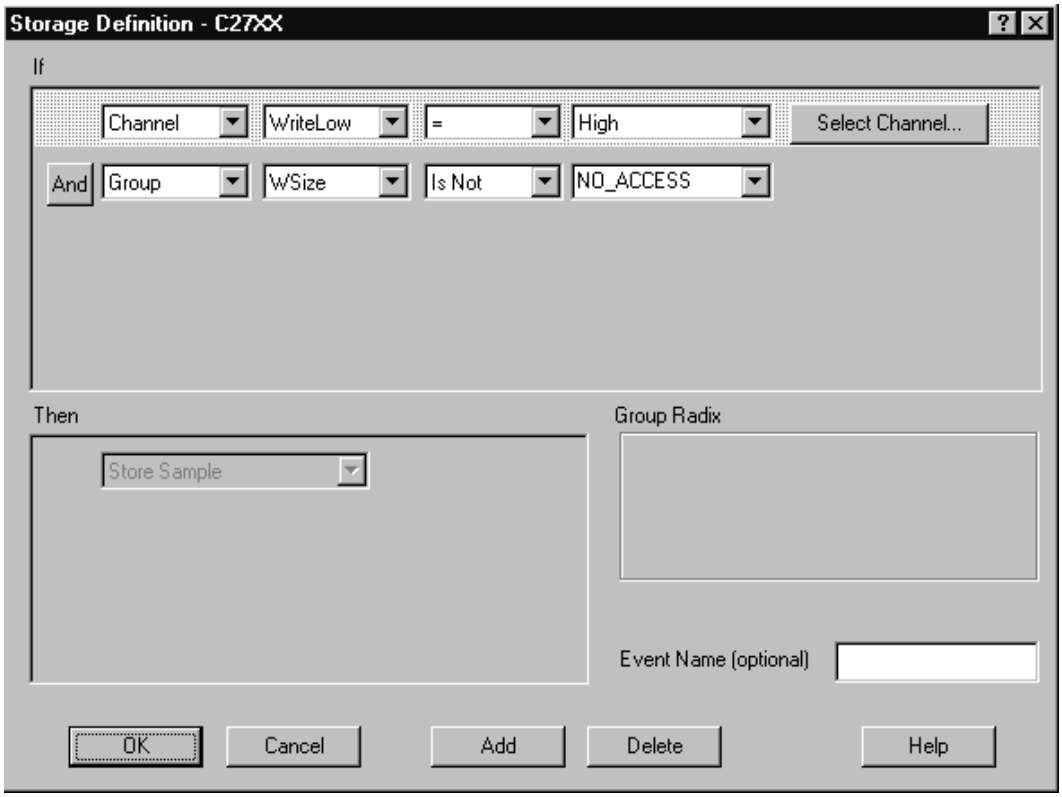
8) From the first pulldown menu, select *Group*.

9) From the Second pulldown menu, select *WSize*.

10) From the Third pulldown menu, select *IsNot*.

11) From the Fourth pulldown menu, select *NO_ACCESS*. After this step, the Storage Definition window should look similar to Figure 3–11.

Figure 3–11. The Storage Definition Window



Creating States

1) Click the *State 1* button to display the *State 1* box on the right-hand side of Trigger window.

2) In the State 1 box, click If to display the Clause Definition window.

3) In the *IF* area, select *Channel* from the first pulldown menu.

4) In the *Then* area, select *Trigger* from the first pulldown menu. After this step, the Trigger window that displays the entire trace algorithm should look similar to Figure 3–13.

5) Click OK to accept the clause definitions.

Figure 3–12. The Clause Definition Window

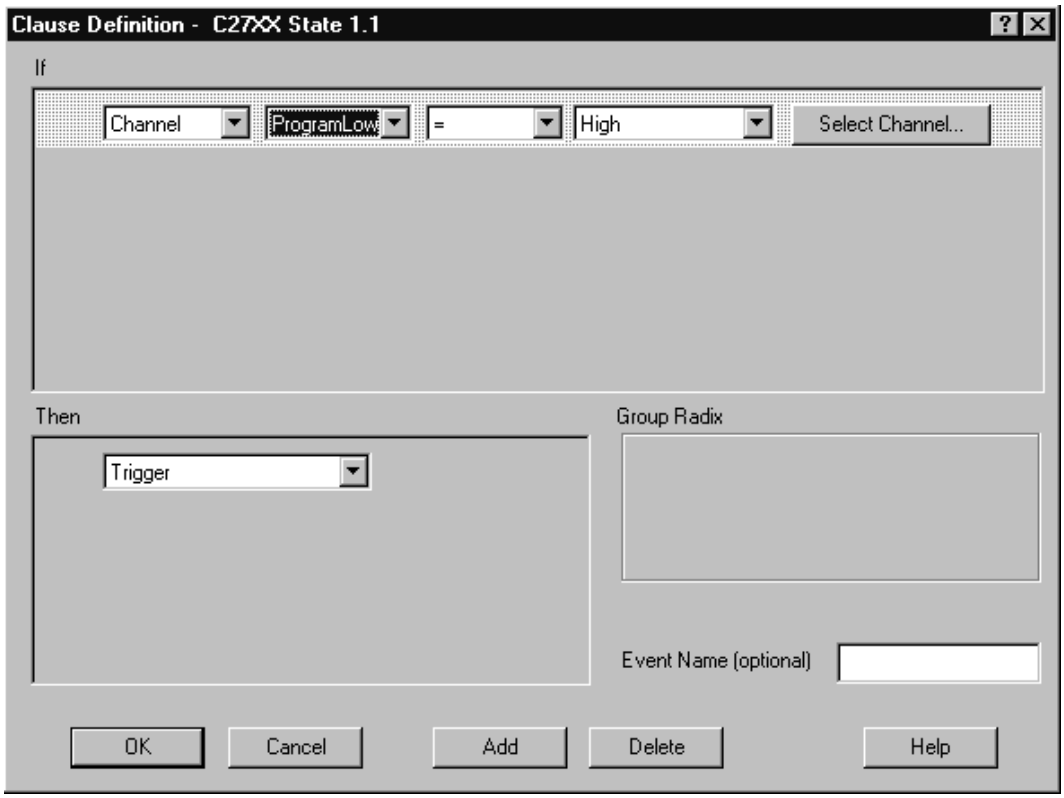
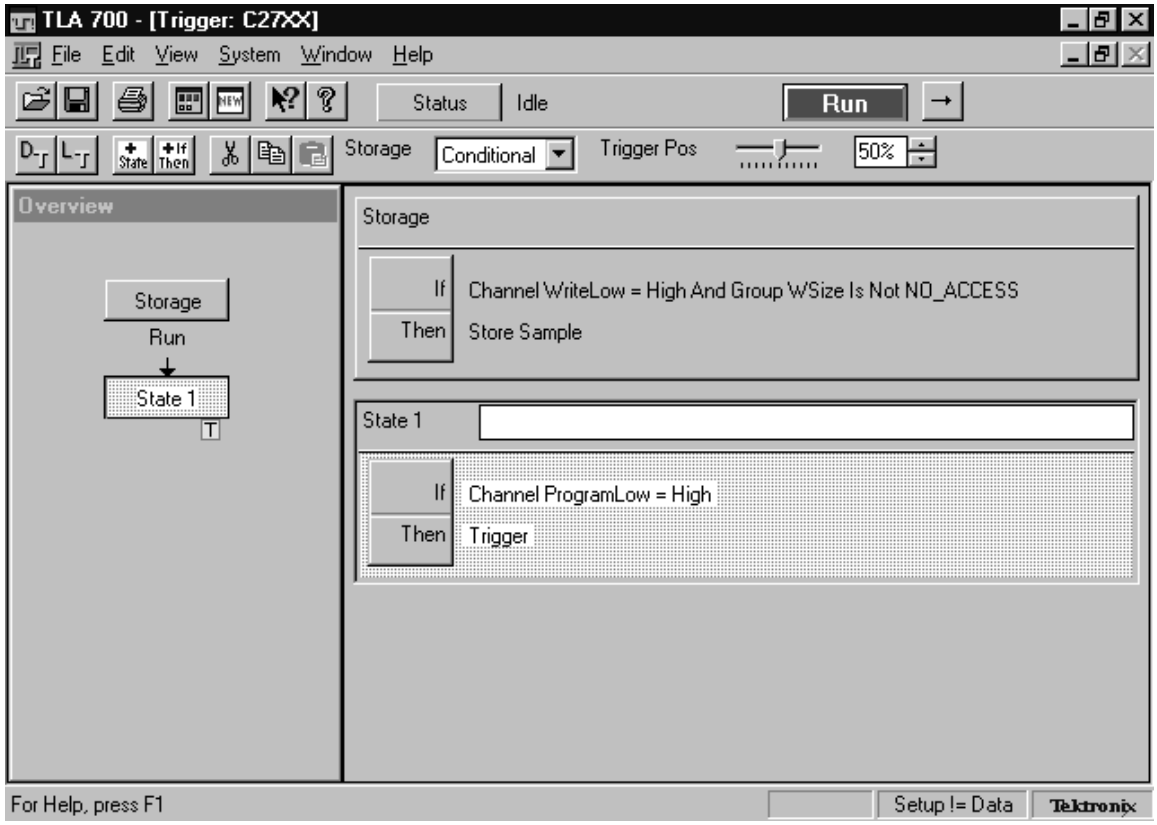


Figure 3–13. Trigger Window Display With Trace Capture Algorithm



3.4.3 Using the TLA704 and the ’C27x Debugger to Acquire Trace Data

To execute source code on your target and acquire the trace data, you must use the ’C27x source code debugger with the TLA704. The debugger runs the code on the target and the TLA704 acquires the data.

To acquire trace data, follow these steps:

1) Click RUN from any TLA704 window.

2) From the command-line of the ’C27x debugger, type **run** and press enter.

This procedure causes the source code to run on the target. When the data on the target matches the conditions programmed into range logic, the range logic outputs a signal that triggers the TLA704 to acquire and store the target data associated with the range logic signal.

3.4.4 Viewing the Disassembly of Trace Capture Data

Once the TLA704 has completed capturing trace data, the data is disassembled for display. To view the disassembled trace data, open the TLA704’s *Listing* window from the TLA704 main window. Your trace data display should look similar to the following figures:

Figure 3–14. Disassembly Display Showing Trace Data For Main Function Entry Point

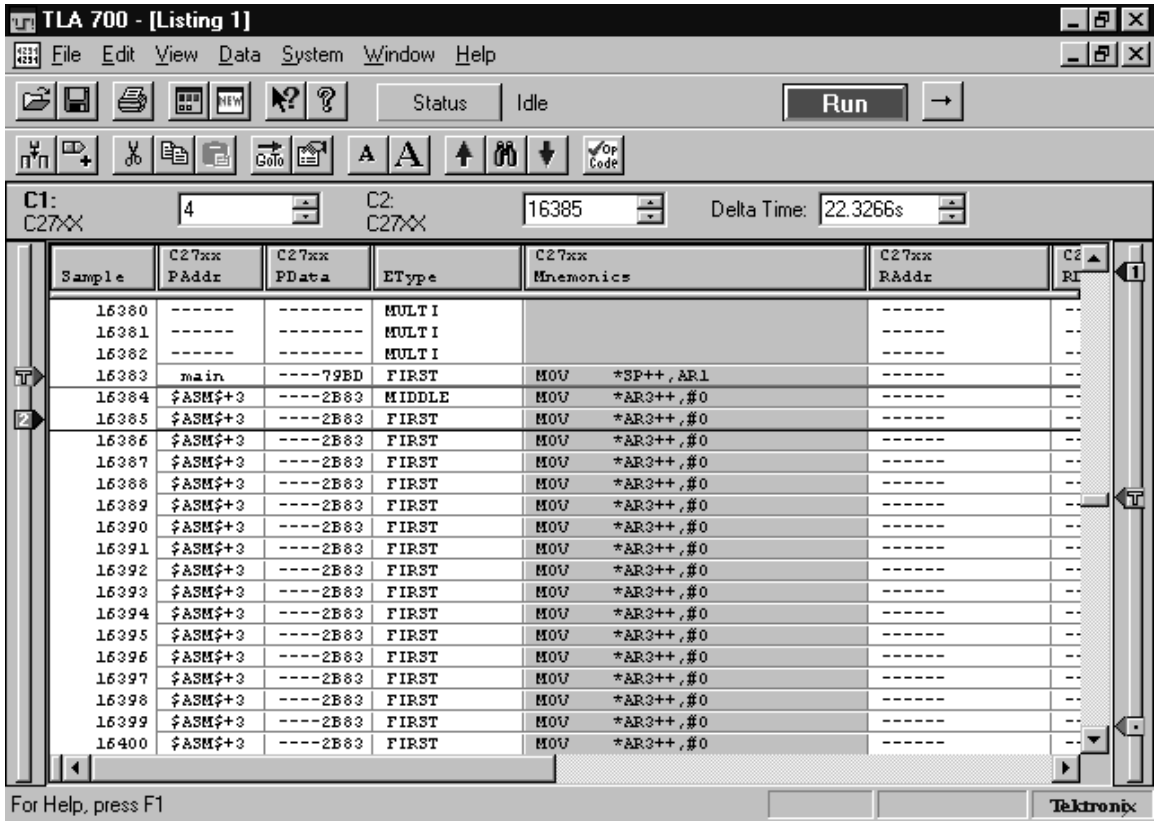
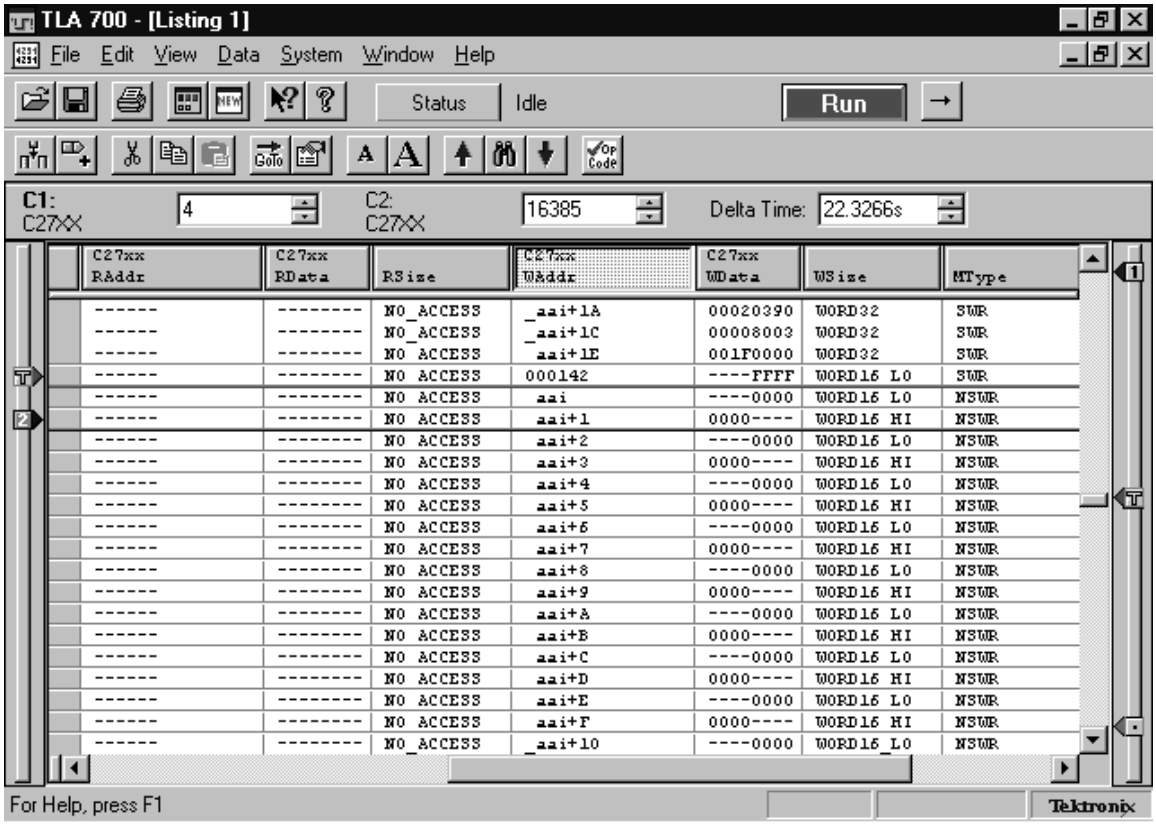


Figure 3–15. Disassembly Display Showing Trace Data For AAI Array



3.4.5 Saving Your Trace Data

Once you have acquired your trace data, you can save it as a separate file. To save your trace data file, follow these steps:

1) From the File menu within the TLA704 main window, select *Save System As* to display the Save As dialog box.

2) On the Save As dialog box, select the following path on your Host PC where C:\ is the name of your hard drive:

   `C:\C27xDSP\EMU\Tutorial`

3) In the File Name dialog box, enter *MyAlgorithm*.

4) Click *Save* to save your algorithm and trace data.

3.4.6 Getting More Information

For more information about the TLA704 acquisition software, see the online help on your TLA704 and your Tektronix manuals.

For more information about the BTT application, see Chapter 4 in this manual.

For more information about the 'C27x microprocessor support package, see the *TMS320C27x XDS512RL Support Package User's Guide* provided by Tektronix, Inc.

For more information about the 'C27x C source debugger and code generation tools, see the following manuals:

- *TMS320C27x C Source Debugger User's Guide*
- *TMS320C27x Optimizing C Compiler User's Guide*
- *TMS320C27x Assembly Language Tools User's Guide*

Chapter 4

Using the BTT Application

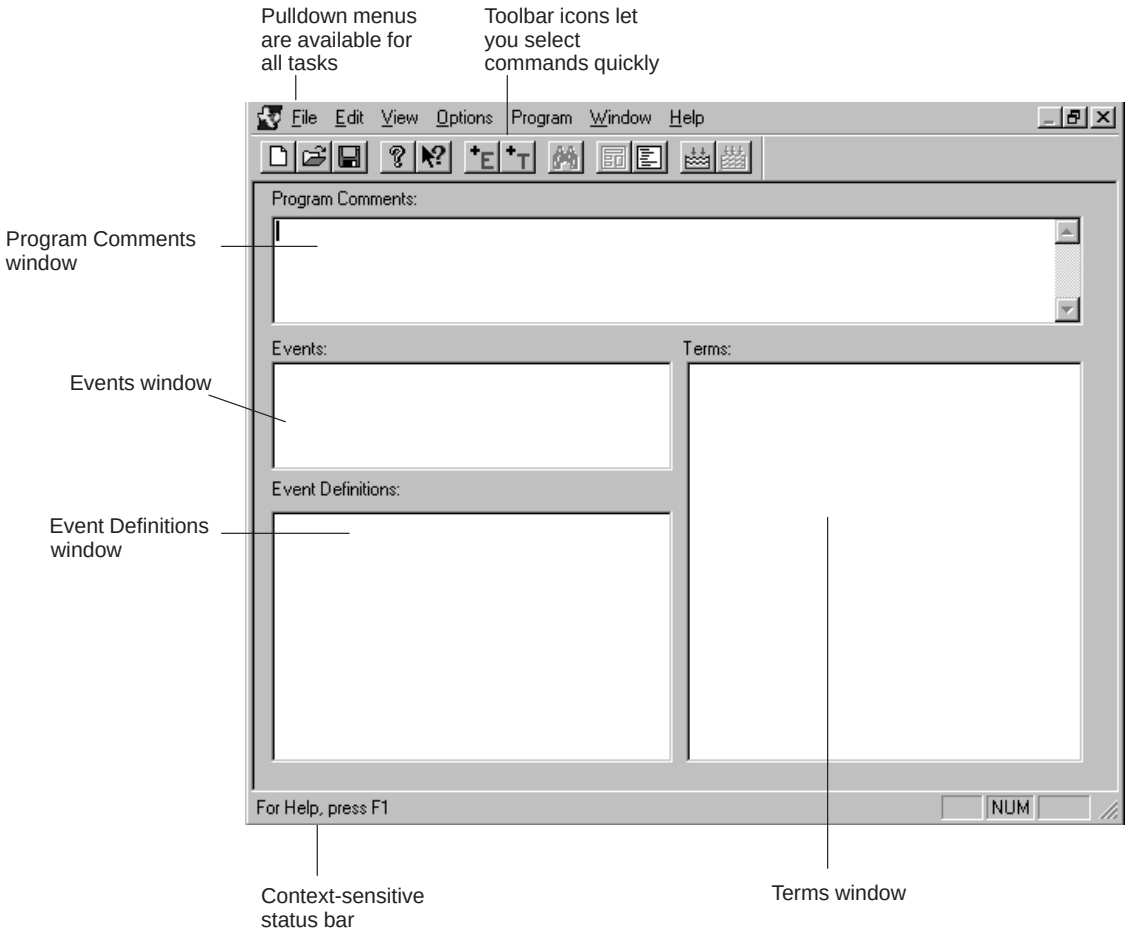
This chapter explains what the BTT application does and how to use it.

Topic **Page**

4.1 Description of the BTT Application

The BTT application is software for creating the trace language programs that you program into the range logic on the XDS512RL emulator. Figure 4–1 shows the parts of the BTT application.

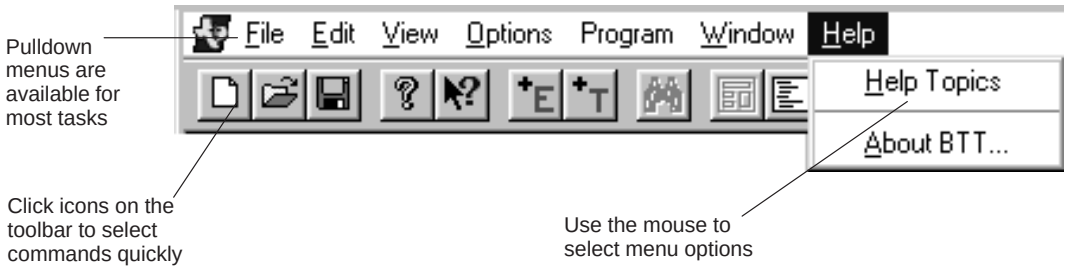
Figure 4–1. The BTT Application Form View



4.2 Using the Menubar and the Toolbar

At the top of the BTT window are the menu bar and the toolbar. The menu selections and the toolbar offer you several methods for entering commands. Figure 4–2 illustrates the basic menu bar and the toolbar.

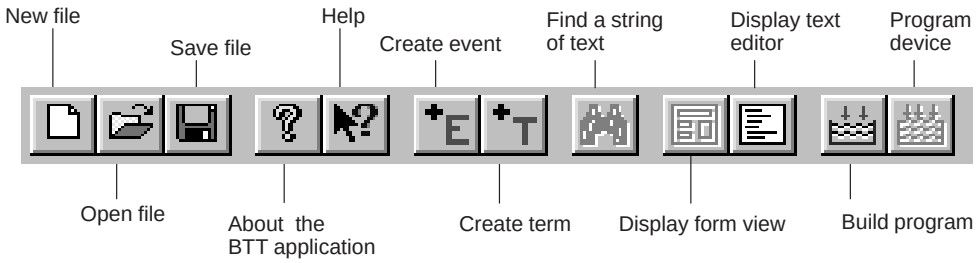
Figure 4–2. The Menu Bar and the Toolbar



4.2.1 Overview of the toolbar

The toolbar provides you with quick access to commonly used commands. Figure 4–3 shows the commands that you can execute from the toolbar.

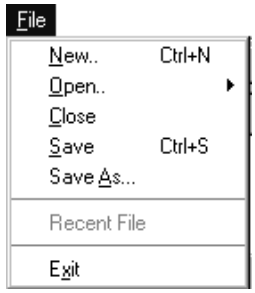
Figure 4–3. The Toolbar



4.2.2 Summary of the BTT Menus

The following figures show each of the BTT menus and briefly describe the tasks you can perform with each menu option. The individual menu options are discussed in the various sections throughout this book as they apply to the topic that is being discussed.

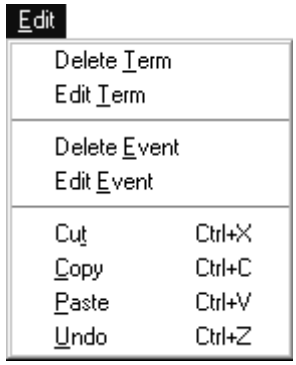
Figure 4–4. The File Menu



The File menu allows you to:

- Create new trace language files
- Open existing trace language files or source files
- Close trace language files
- Save the trace language program with the current name
- Save the trace language program with a new name
- Open a file from a list of the most recently accessed files
- Exit the BTT application

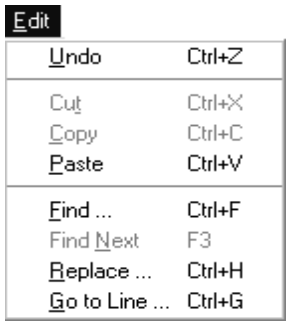
Figure 4–5. (a) Edit Menu in Form View



In the Form View, the Edit menu allows you to:

- Delete selected term
- Edit selected term
- Delete selected event
- Edit selected event
- Cut text selected in the Program Comments window
- Copy text selected in the Program Comments window
- Paste text into the Program Comments window
- Undo the last edit made to text in the Program Comments window

Figure 4–5. (b) Edit Menu in Text Editor



In the Text Editor, the Edit menu allows you to:

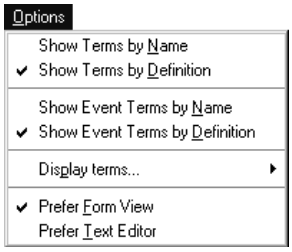
- ❑ Undo previous command(s)
- ❑ Cut selected items
- ❑ Copy selected items
- ❑ Paste selected items
- ❑ Search a file for a string of text
- ❑ Find the next occurrence of a search string
- ❑ Search for and replace text with specified information
- ❑ Go to a specific line number within the trace language file

Figure 4–6. View Menu

The View menu allows you to:

- ❑ Display the Text Editor or the Form View immediately
- ❑ Hide or show the toolbar, status bar, error listing, or event comments

Figure 4–7. (a) Options Menu in Form View



In the Form View, the Options menu allows you to:

- ❑ Show terms in the term window by name or definition
- ❑ Show terms in the event definition window by name or definition
- ❑ Determine what type of terms are displayed in the Terms window
- ❑ Set default view preference (form view or text editor). The BTT application displays the default view when you invoke it or when you open a new file.

Figure 4–7. (b) Options Menu in Text Editor

In the Text Editor, the Options menu allows you to:

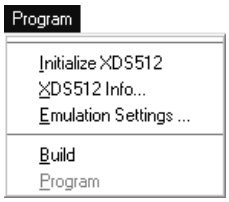
- ❏ Set default view (form view or text editor)

Figure 4–8. (a) Program Menu in Form View

In the Form View, the Program menu allows you to:

- ❏ Create an event
- ❏ Create a term
- ❏ Initialize the XDS512RL
- ❏ Check software and hardware version numbers
- ❏ Change emulator settings
- ❏ Turn grouping on/off
- ❏ Build trace language programs
- ❏ Program the range logic

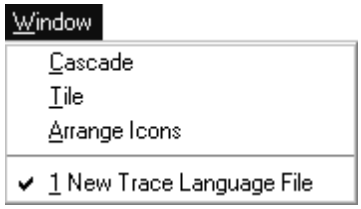
Figure 4–8. (b) Program Menu in Text Editor



In the Text Editor, the Program menu allows you to:

- ❏ Initialize the XDS512RL
- ❏ Check software and hardware version numbers
- ❏ Change emulator settings
- ❏ Build trace language programs
- ❏ Program the range logic

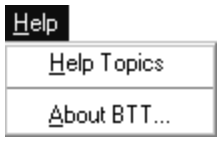
Figure 4–9. Window Menu



The Window menu allows you to:

- Arrange the BTT application windows
- Make an individual window active

Figure 4–10. Help Menu



The Help menu allows you to:

- Access the BTT application help topics
- Access copyright and version information about the BTT applicat

4.2.3 Selecting Menu Options

To open a menu and choose a menu option, follow these steps:

1) Click the name of the menu that you want to open. This opens the menu.

2) Click the menu option that you want to use.

You can also open a menu by using the ALT key. Press ALT then type the letter that is underlined in the menu name. To select a menu option from an open menu, type the letter that is underlined in the option name.

4.2.4 Understanding the Menu Conventions

The BTT application uses the menu conventions listed in Table 4–1.

Table 4–1. Menu Conventions

Menu Convention	Meaning
A menu or menu option is dimmed or not visible	You cannot access this menu or menu option currently. For example, if the BTT application cannot initialize the XDS512RL, then the Program button on the Toolbar and the Program command on the Program menu are grayed out.
An ellipsis (...) follows a menu option	When you choose this menu option, a dialog box is displayed in which you must supply or confirm some information.
A check mark is next to a menu option	The menu option is in effect. For example, when the toolbar is displayed, you see a check mark next to the Toolbar option on the View menu.
A key combination is next to a menu option	You can use the key combination to select the menu option (without having to open the menu).

4.3 The BTT Application Windows

The BTT application contains the following windows:

- ❏ Form view main window
- ❏ Text editor
- ❏ Source view windows

This section explains the BTT application windows.

4.3.1 The Form View Main Window

The form view main window contains the following windows:

- ❏ Events window
- ❏ Terms window
- ❏ Event definitions window
- ❏ Program comments window
- ❏ Event comments dialog box
- ❏ Error list dialog box

Events Window

The Events window displays your events. By right-clicking on the Events window, you can display a context-sensitive menu that allows you to perform the following actions:

Menu Option	Action
Create Event	Displays the Event Editor so you can create an event. For more information about creating events, see page 4-26.
Delete Event	Deletes the selected event.
Edit Event	Displays the Event Editor, which contains the selected event's definition. You can also edit an event by double-clicking on it in the Events window.
Duplicate Event	Copies the event for you to save with a new name.

Terms window

The Terms window displays your terms. By right-clicking on the Terms window, you can display a context-sensitive menu that allows you to perform the following actions:

Menu Option	Action
Create Term	Displays the Term Editor so you can define a term. For more information about defining terms, see page 4-16.
Delete Term	Deletes the selected term and removes it from its assigned events.
Edit Term	Displays the Term Editor, which contains the selected term's definition. You can also edit a term by double-clicking on it in the Terms window.
Assign Term to Event	Assigns the selected term to the selected event. You can also assign a term to an event by dragging and dropping it into the Events Definition window.
Duplicate Term	Copies the term for you to save with a new name.
Display terms	Displays all terms or only terms compatible with a specific bus.
Show terms by Name	Displays terms by their term names.
Show terms by Definition	Displays terms by their definitions. If a term does not have a definition, it is displayed by name.

Event Definitions window

The Events Definition window displays the terms that are assigned to the selected event. By right-clicking on the Events Definition window, you can display a context-sensitive menu that allows you to perform the following actions:

Menu Option	Action
Remove from Event	Removes the selected term from the event definition.
Edit the Term	Displays the Term Editor, which contains the selected term's definition. You can also edit a term by double-clicking on it.
Show Terms by Name	Displays terms by their term names.
Show terms by Definition	Displays terms by their definitions.

Program Comments Window

The Program Comments window provides space for you to enter comments about your trace language program. To enter comments, click in the Program Comments window and begin typing. By right-clicking on the Program Comments window, you can display a context-sensitive menu that allows you to perform the following actions. To use the Cut, Copy, or Delete menu options, you must first select text by dragging the cursor over it or by choosing the Select All menu option.

Menu Option	Action
Undo	Undoes the last edit made to text in the window
Cut	Removes the selected text from the window and stores the text. The stored text can be placed elsewhere in this window or in another text window using a Paste menu option.
Copy	Stores a copy of the selected text. The stored text can be placed elsewhere in this window or in another text window using a Paste menu option.
Paste	Inserts the text that was last stored using a Cut or Copy menu option. The text is inserted at the current position of the cursor.
Delete	Removes the selected text from the window. The text is not stored; it cannot be recovered.
Select All	Selects all of the text in the window

Event Comments Dialog Box

To display the Event Comments dialog box shown in Figure 4–11, select Event Comment from the View menu. The Event Comments dialog box displays the comments for a selected event in the Events window. You can use the Event Comments dialog box as a short-cut for changing event comments by clicking in the box and typing the new comment information. The information in the Event Comments dialog box updates when you select a new event in the Events window. The Event Comments dialog box closes when you open the Event Editor.

Figure 4–11. The Event Comments Dialog Box

Error List Dialog Box

The Error List dialog box shown in Figure 4–12 displays messages about errors in your term and event definitions as well as communication errors between the BTT application and the XDS512RL emulator board. In the Form View, double-clicking on an event or term error displays the appropriate editor for you to correct the error. In the Text Editor, double-clicking on an event or term error takes you to the beginning of the event or term definition within the trace language program.

Figure 4–12. The Error List Dialog Box

4.3.2 The Text Editor

To display the text editor, click  from the toolbar or select Text Editor from the View menu. The text editor allows you to view and edit you trace language programs. If you are editing any term or event definitions with the Term Editor or Event Editor, you must save or cancel your changes before you can display the text editor. If contents in the Term Editor or Event Editor are unchanged, they are closed before switching to the text editor. In addition to standard features such as cut, copy, paste, and undo, the text editor also provides the following commands from the Edit menu:

- **Find**: The Find command lets you search for a string of text within a trace language file or a source file. You can invoke the Find command in the following ways:
  - Click  from the toolbar
  - Select Find from the Edit menu
  - Press Ctrl + F

The Find command displays the Find dialog box shown in Figure 4–13. The Find dialog box lets you perform case-sensitive searches for strings of text. The Up and Down radio buttons allow you to search either up or down from the position of the cursor.

Figure 4–13. The Find Dialog Box

- **Find Next**: The Find Next command searches for the next occurrence of the last text string you searched for in a trace language file. The Find Next command is dependent on the Up and Down radio buttons in the Find dialog box. You can invoke the Find Next command in the following ways:
  - Select Find Next from the Edit menu
  - Press F3 on your keyboard
  - Press the Find Next button in the Find dialog box

- **Replace**: The Replace command displays the Replace dialog box shown in Figure 4–14. The Replace dialog box lets you search for a string of text that you specify in the Find what field and replace it with the string of text

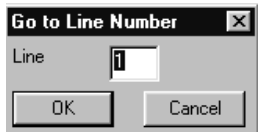
you specify in the Replace with field. You can invoke the Find Next command in the following ways:

- Select Replace from the Edit menu
- Press Ctrl + H on your keyboard

Figure 4–14. The Replace Dialog Box

- **Go to line**: The Go to Line command displays the Go to Line Number dialog box shown in Figure 4–15. The Go to Line command lets you jump to a specific line number within the current trace language file. You can invoke the Go to Line command in the following ways:
  - Select Go to Line from the Edit menu
  - Press Ctrl + G on your keyboard

Figure 4–15. The Go To Line Number Dialog Box



4.3.3 The Source View Window

You can display the source view window by selecting Open/Source File from the File menu or by selecting *View the source file* from the context-menu in the Address Editor's Program/Data window. This command displays the symbol's source file in a Source View window. The Source View window lets you browse the source file and search it for a string of text. This window is for viewing only, you cannot edit files in the Source View Window. You can display up to ten Source View windows at the same time.

4.4 Creating Terms With the Term Editor

The Term Editor lets you create and edit terms. A term compares an address bus's current value to one or more addresses or address ranges that you specify and/or checks for a CPU condition or conditions that you specify. A term definition includes the following information:

Term name	A descriptive name that you assign to a term. Term names can be alphanumeric, can contain underlines, are case sensitive, and must start with a letter. A term cannot have the same name as another term, event, or qualifier.
Bus	Choose the bus type you want associated with the term. Table 4–2 lists the choices for the Bus field and explains the effects of choosing each bus type.
Address List	Displays the address expression associated with the term.
Qualifiers	Choose from a list of bus qualifiers to append to the term definition. For descriptions of the qualifiers, see page 4-21.
Qualifier Operators	The available operators are as follows: (Opening parenthesis) Closing parenthesis && AND || OR
Comments	Any notes you want to include about the term.
Qualifier Expression	Enter a qualifier expression by dragging and dropping qualifiers and qualifier operators into this field. You can also add qualifiers and qualifier operators to a qualifier expression by double-clicking on them from the list-boxes. You can reposition qualifier items within an expression by dragging them from one position within the box and dropping them in another position within the box. You can remove an item by dragging it outside the Qualifier Expression field. This field contains a context menu that lets you remove the selected object or clear the entire field.

You can display the Term Editor in the following ways:

- Clicking  on the menu bar
- Double-clicking in the Terms window
- Selecting Create Term from the context menu in the Terms window
- Selecting Edit Term from the context menu in the term list or the event definition list

The term editor is modeless, so you can open other windows within the BTT application while you are using the Term Editor.

Table 4–2. Choosing A Bus Value

Bus	Possible Form(s) For Term†	Available Qualifiers‡	Term Can Be Assigned To These Events§
Any	❑ Qualifier expression	Qualifiers that apply to any program-space or data-space activity	All events
PA	❑ Address expression (using PA bus) ❑ Qualifier expression ❑ Address expression && qualifier expression	Qualifiers that apply to program-space activity	Events that trigger program signals, such as ProgramLow or Program5
RA	❑ Address expression (using RA bus) ❑ Qualifier expression ❑ Address expression && qualifier expression	Qualifiers that apply to read activity in data space	Events that trigger read signals, such as ReadLow or Read5
WA	❑ Address expression (using WA bus) ❑ Qualifier expression ❑ Address expression && qualifier expression	Qualifiers that apply to write activity in data space	Events that trigger write signals, such as WriteLow or Write5
R/W	❑ Qualifier expression	Qualifiers that apply to read or write activity in data space	Events that trigger read signals or write signals

† An address expression defines one or more addresses or address ranges that a term uses when monitoring an address bus. A qualifier expression defines one or more CPU conditions that a term monitors.
‡ Qualifiers are described in section 4.5 on page 4-21.
§ Events are described in section 4.6, *Creating Events With the Event Editor*, on page 4-26.

Figure 4–16. Term Editor

4.4.1 Creating terms

To create a term, follow these steps:

1) From the toolbar, click  to display the Term Editor dialog box.

2) In the Term Name field, enter a descriptive name for the term. Term names can be alphanumeric, can contain underlines, are case-sensitive, and must start with a letter. A term cannot have the same name as another term, event, or qualifier.

3) If you want the term to monitor a specific bus or specific bus qualifiers, right-click in the Bus field to select a bus. Selecting a bus displays the selected bus's qualifiers in the Program/Data Qualifiers window.

4) In the comments field, enter any comments you want to add.

5) If you want to monitor an address bus, click Add... to display the Address Editor and enter the address or address range that you want the term to monitor. For more information about using the Address Editor, see section 4.4.2.

6) If you want to add qualifiers to the term definition, drag the desired combination of qualifiers and qualifier operators into the *Qualifier Expression* field. Qualifier operators follow ANSI C precedence. To remove a qualifier or qualifier operator from the *Qualifier Expression* field, drag it out of the field. For more information about the qualifiers, see section 4.5 on page 4-21.

7) Click OK to save the term definition. After you click OK, the term displays in the Terms window.

The Terms window supports context menus. You can right-click in the Terms window to display a context menu with the choices described in section 4.3.1, *Terms Window*, on page 4-10.

4.4.2 Address Editor

Use the Address Editor to define the address or address range that you want the term to monitor. To display the Address Editor, select Add... from the Term Editor. The Address Editor is shown in Figure 4–17 and contains the following fields:

Address or Range box — Select the *Address* or *Range* radio button to define an address or address range for the term.

Address — Choose the *Address* radio button to define an absolute address for the term to monitor. An address definition can contain the following values:

- **Symbol**: Drag a symbol from the Program/Data window. You can also type information into this field.
- **+/–**: Choose + or – from the drop-list.
- **Constant**: Enter a decimal or hexadecimal number. You can also type information into this field.

To define an absolute address, enter an absolute address in the *Constant* field and leave the *Symbol* field blank. The BTT application ignored the *+/–* field.

Start Address/ End Address (Range) — Choose the *Range* radio button to define an address range for the term to monitor. A range is defined by a start address and an end address.

The *Start Address/End Address* fields define the start and end addresses of an address range and contain the following information:

- **Symbol**: Drag a symbol from the Program/Data window.
- **+/–**: Choose + or – from the drop-list.
- **Constant**: Enter a decimal or hexadecimal number.
- **Length**: Check the *Length* box to specify a length as the end of a range. Length values are inclusive.

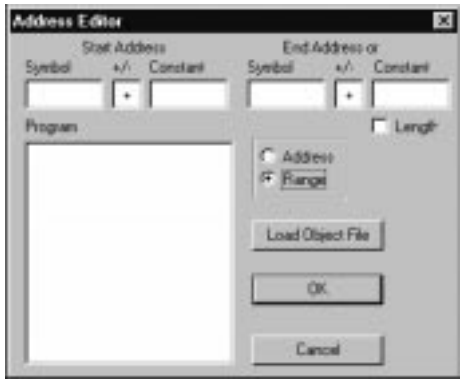
Program/Data Window — The Program/Data window displays the program or data symbols of a target executable file. If the address or range is in program space, the word Program is above the window; if the address or range is in data space, the word Data is above the window.

Load Object File — Use this button to populate the Program/Data window with the program or data symbols from a target executable file.

Note:

The Program/Data field only displays global program/ Data symbols.

Figure 4–17. Address Editor



The Program/Data Window

The Address Editor Program/Data window contains a context menu. By right-clicking on the Program/Data window, you can display a context-sensitive menu that allows you to perform the following actions:

Menu option	Action
Properties	Displays the following information about the selected symbol: ❑ Which source file contains the symbol definition ❑ Which lines in the source file contain the symbol definition. ❑ Symbol address
Find Symbol	Search the Program/Data window for a string.
Sort Alphabetically	Displays the symbols in the Program/Data window alphabetically
Sort list by address	Sorts symbols by address
View Source File	Lets you open and view the selected symbol's source file.
Display Runtime Symbols	Displays runtime symbols in the Program/Data window
Do not Display Runtime Symbols	Removes runtime symbols from the Program/Data window display.

4.5 Qualifiers

The Term Editor (see section 4.4 on page 4-16) allows you to use qualifiers for each term definition. These qualifiers allow you to make your terms partly or solely dependent on particular conditions in the CPU. For example, suppose you have the following term:

PA = = 0x100

The term is composed entirely of an address expression. This expression makes the term true if the address 0x100 appears on the program address bus. Now suppose you want to add two CPU conditions. In addition to meeting the bus condition PA = = 0x100, you want the term to be true only if the CPU is not servicing a high-priority interrupt (NotHPI) *and* the CPU has decoded an interrupt (IntJam) or a conditional branch (IsBranch). The Term Editor allows you to add these other conditions by adding a qualifier expression to your term:

PA = = 0x100 && NotHPI && (IntJam || IsBranch)

You can use the qualifier operators as follows:

Qualifier operator	Definition
&&	Specifies an AND operation. One of these operators is automatically added between the address expression and the qualifier expression.
||	Specifies an OR operation.
( )	Indicate a subexpression that must be evaluated before the rest of the expression is evaluated.

You can assign qualifiers and qualifier operators to term definitions by dragging and dropping them into the Qualifier Expression field or by double-clicking on them in the drop-lists to add them to the end of the qualifier expression.

In the previous two examples, address expression were given. However, a term can be composed entirely of a qualifier expression. Suppose you choose Bus = WA and you want a term that is true if PAGE0 stack addressing mode is selected. Your term would appear as follows:

PageZeroStack

If you choose Any or R/W for the Bus field in the Term Editor, your term can only contain a qualifier expression; in these cases, an address expression is not allowed. However, you can relate these terms to address buses when you assign them to events. A term with Bus = Any can be assigned to an event associated with the program address bus, the data-read address bus, or the data-write address bus. A term with Bus = R/W can be assigned to an event associated with the data-read address bus or the data-write address bus.

The qualifiers available for a given term depend on the value in the Bus field:

For Information About These Qualifiers...	See...
Qualifiers available when Bus = Any	Section 4.5.1 on page 4-22
Qualifiers available when Bus = PA	Section 4.5.2 on page 4-22
Qualifiers available when Bus = RA or Bus = WA	Section 4.5.3 on page 4-23
Qualifiers available when Bus = R/W	Section 4.5.4 on page 4-25

4.5.1 Qualifiers Available When Bus = Any

When the Bus field of the Term Editor contains Any, only the qualifiers in Table 4–3 are available. These are the qualifiers that apply to any of the three buses (program, read, and write).

Table 4–3. Qualifiers Available When Bus = Any

Qualifier	Description
HPI	The ’C27x is processing a high-priority interrupt.
NotHPI	The ’C27x is not processing a high-priority interrupt.

4.5.2 Qualifiers Available When Bus = PA

When the Bus field of the Term Editor contains PA, you can use any of the qualifiers described in Table 4–4. Do not use the qualifier expressions listed in Table 4–5.

Table 4–4. Qualifiers Available When Program Bus is Monitored (Bus = PA)

Qualifier	Description
HPI	The ’C27x is processing a high-priority interrupt.
NotHPI	The ’C27x is not processing a high-priority interrupt.
Execute	The ’C27x is executing an instruction.
IntJam	The ’C27x has decoded an interrupt.
NotIntJam	The ’C27x has not decoded an interrupt.
First	The ’C27x is executing the first instruction after a discontinuity.
Last	The ’C27x is executing the last instruction before a discontinuity.
Middle	The instruction being executed is neither the last instruction before a discontinuity nor the first instruction after a discontinuity.

Table 4–4. Qualifiers Available When Program Bus is Monitored (Bus = PA)(Continued)

Qualifier	Description
IsBranch	The ’C27x is executing a conditional branch.
IsNotBranch	The ’C27x is not executing a conditional branch.
TrueBranch	The ’C27x has executed a conditional branch whose condition was true.
FalseBranch	The ’C27x has executed a conditional branch whose condition was false.
IndirectOrCall	The ’C27x is executing a branch or call instruction that is using indirect addressing.
PipeStall	The ’C27x pipeline is stalled for pipeline protection.

Table 4–5. Illegal Qualifier Expressions When Bus = PA

- IsBranch && IsNotBranch
- IsNotBranch && TrueBranch
- IsNotBranch && FalseBranch
- IsNotBranch && IndirectOrCall
- TrueBranch && FalseBranch
- TrueBranch && IndirectOrCall
- TrueBranch && IndirectOrCall

4.5.3 Qualifiers Available When Bus = RA or Bus = WA

When the Bus field of the Term Editor contains RA, the data-read address bus is monitored. When the Bus field contains WA, the data-write address bus is monitored. Table 4–6 describes the qualifiers for both cases because they are nearly identical. The difference is seen in qualifiers that correspond to memory accesses. For Bus = RA, these qualifiers end with Read (for example, AnyRead); for Bus = WA, they end with Write (for example, AnyWrite). You can connect multiple qualifiers in a qualifier expression, but do not use the types of qualifier expressions listed in Table 4–7 (page 4-25).

Table 4–6. Qualifiers Available When Read Bus or Write Bus is Monitored (Bus = RA or Bus = WA)

Qualifier	Description
HPI	The ’C27x is processing a high-priority interrupt.
NotHPI	The ’C27x is not processing a high-priority interrupt.
PageZero	PAGE0 direct addressing mode is selected.
NotPageZero	PAGE0 stack addressing mode is selected.
AnyRead/Write	The ’C27x is reading from/writing to memory.
NoAccessRead/Write	The ’C27x is not reading from/writing to memory.

Table 4–6. Qualifiers Available When Read Bus or Write Bus is Monitored (Bus = RA or Bus = WA)(Continued)

Qualifier	Description
AnyProgramRead/Write	The ’C27x is reading from/writing to program space.
AnyStackRead/Write	The ’C27x is reading from/writing to the stack.
AnyNonStackRead/Write	The ’C27x is reading from/writing to memory but is not accessing the stack.
Any16BitProgramRead/Write	The ’C27x is reading/writing a 16-bit value from/to program space.
Even16BitProgramRead/Write	The ’C27x is reading/writing a 16-bit program-space value at an even address.
Odd16BitProgramRead/Write	The ’C27x is reading/writing a 16-bit program-space value at an odd address.
A32BitStackRead/Write	The ’C27x is reading/writing a 32-bit value from/to the stack.
Any16BitStackRead/Write	The ’C27x is reading/writing a 16-bit value from/to the stack.
Even16BitStackRead/Write	The ’C27x is performing a 16-bit stack read/write at an even address.
Odd16BitStackRead/Write	The ’C27x is performing a 16-bit stack read/write at an odd address.
AnyLowByteStackRead/Write	The ’C27x is reading from/writing to the low byte of a stack value.
EvenLowByteStackRead/Write	The ’C27x is reading from/writing to the low byte of a stack value at an even address.
OddLowByteStackRead/Write	The ’C27x is reading from/writing to the low byte of a stack value at an odd address.
AnyHighByteStackRead/Write	The ’C27x is reading from/writing to the high byte of a stack value.
EvenHighByteStackRead/Write	The ’C27x is reading from/writing to the high byte of a stack value at an even address.
OddHighByteStackRead/Write	The ’C27x is reading from/writing to the high byte of a stack value at an odd address.
Any16BitNonStackRead/Write	The ’C27x is reading/writing a 16-bit value but is not accessing the stack.
Even16BitNonStackRead/Write	The ’C27x is performing a 16-bit, non-stack read/write at an even address.
Odd16BitNonStackRead/Write	The ’C27x is performing a 16-bit, non-stack read/write at an odd address.
A32BitNonStackRead/Write	The ’C27x is reading/writing a 32-bit value but is not accessing the stack.

Table 4–6. Qualifiers Available When Read Bus or Write Bus is Monitored (Bus = RA or Bus = WA)(Continued)

Qualifier	Description
AnyLowByteNonStackRead/Write	The ’C27x is reading from/writing to the low byte of a non-stack data-space value.
EvenLowByteNonStackRead/Write	The ’C27x is performing a low-byte, non-stack read/write at an even address.
OddLowByteNonStackRead/Write	The ’C27x is performing a low-byte, non-stack read/write at an odd address.
AnyHighByteNonStackRead/Write	The ’C27x is reading from/writing to the high byte of a non-stack data-space value.
EvenHighByteNonStackRead/Write	The ’C27x is performing a high-byte, non-stack read/write at an even address.
OddHighByteNonStackRead/Write	The ’C27x is performing a high-byte, non-stack read/write at an odd address.

Table 4–7. Illegal Qualifier Expression Types When Bus = RA or Bus = WA

- Even...&&Odd...
- NoAccessRead && ...Program...
- NoAccessRead && ...NonStack...
- ...Program... && ...Stack...
- NoAccessRead && ...Stack...
- ...Program... && ...NonStack...
- ...Stack... && ...NonStack...

4.5.4 Qualifiers Available When Bus = R/W

When the Bus field of the Term Editor contains R/W, only the qualifiers in Table 4–8 are available. These are the qualifiers that apply to both the data-read address bus and the data-write address bus.

Table 4–8. Qualifiers Available When Bus = R/W

Qualifier	Description
HPI	The ’C27x is processing a high-priority interrupt.
NotHPI	The ’C27x is not processing a high-priority interrupt.
PageZero	PAGE0 direct addressing mode is selected.
NotPageZero	PAGE0 stack addressing mode is selected.

4.6 Creating Events With the Event Editor

An event is a group of ORed terms that tells the range logic when to output a specific signal. An event is true when the data on the target matches any one of the term definitions within the event. When an event is true, the range logic outputs the event's assigned range logic signal to the external trace equipment. The BTT application has an event editor that lets you create and edit events. An event definition includes the following information:

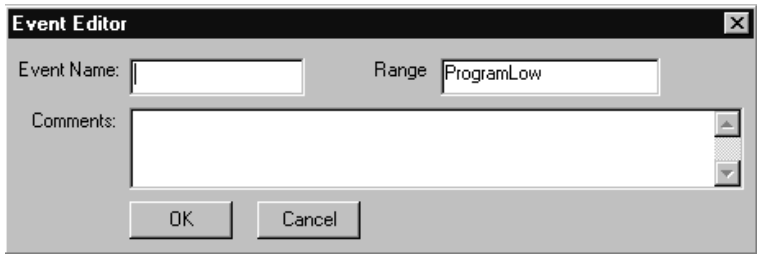
Event Name	An event name is a descriptive name that you assign to an event. Event names can be alphanumeric, are case-sensitive, can contain underlines, and must start with a letter. An event cannot have the same name as another event, term, or qualifier.
Range	The range value is a specific signal that the event tells the range logic to output when the event is true.
Comments	Any comments you want to include about the event.

You can display the Event Editor in the following ways:

- Click  on the menu bar
- Double-click in the Events window
- Select Create Event from the context menu in the Event window

The Event Editor is modeless, so you can open other windows within the BTT application while you are using the Term Editor.

Figure 4–18. The Event Editor



After you create an event, you can assign any of your terms to the event. When you select an event in the BTT application, the terms assigned to that event are displayed in the Event Definitions window.

4.6.1 Range Logic Output Signals

The Range field of an event definition specifies which signal you want the range logic to output when an event is true (when the conditions on the target system match the conditions that you programmed into the range logic). Each of the three range logic banks (program, read, and write) contains eight blocks (numbered 0–7). Blocks 0–3 are tied to ProgramLow, ReadLow, or WriteLow, depending on the chosen bank. Blocks 4–7 are tied the following signals, depending on the chosen bank:

- ❏ A trigger signal (Program4, Program5, Program6, or Program7)
- ❏ ReadHigh and a trigger signal (Read4, Read5, Read6, or Read7)
- ❏ WriteHigh and a trigger signal (Write4, Write5, Write6, or Write7)

Table 4–9, Table 4–10, and Table 4–11 describe the output signals for program-space accesses, data-space reads, and data-space writes, respectively.

Table 4–9. Program Space Range Logic Signals

Signal	Output to...	Output when...
ProgramLow	TLA704	Data on the target system matches the data programmed into the low program space range logic (program space blocks 0–3)
Program4	External probe (JP4–1)	Data on the target system matches the data programmed into block 4 of the program space range logic
Program5	External probe (JP4–3)	Data on the target system matches the data programmed into block 5 of the program space range logic
Program6	External probe (JP4–4)	Data on the target system matches the data programmed into block 6 of the program space range logic
Program7	External probe (JP4–5)	Data on the target system matches the data programmed into block 7 of the program space range logic

Table 4–10. Read Space Range Logic Signals

Signal	Output to...	Output when...
ReadLow	TLA704	Data on the target system matches the data programmed into the low read space range logic (read space blocks 0–3)
ReadHigh	TLA704	Data on the target system matches the data programmed into the high read space range logic (read space blocks 4–7)
Read4	External probe (JP5–1)	Data on the target system matches the data programmed into block 4 of the read space range logic
Read5	External probe (JP5–3)	Data on the target system matches the data programmed into block 5 of the read space range logic
Read6	External probe (JP5–4)	Data on the target system matches the data programmed into block 6 of the read space range logic
Read7	External probe (JP5–5)	Data on the target system matches the data programmed into block 7 of the read space range logic

Table 4–11. Write Space Range Logic Signals

Signal	Output to...	Output when...
WriteLow	TLA704	Data on the target system matches the data programmed into the low write space range logic (write space blocks 0–3)
WriteHigh	TLA704	Data on the target system matches the data programmed into the high write space range logic (write space blocks 4–7)
Write4	External probe (JP6–1)	Output when the data on the target system matches the data programmed into block 4 of the write space range logic
Write5	External probe (JP6–3)	Data on the target system matches the data programmed into block 5 of the write space range logic
Write6	External probe (JP6–4)	Data on the target system matches the data programmed into block 6 of the write space range logic
Write7	External probe (JP6–5)	Data on the target system matches the data programmed into block 7 of the write space range logic

4.6.2 Creating Events

To create an event, follow these steps:

1) From the toolbar, click  to display the Event Editor dialog box.

2) In the Event Name field, enter a descriptive name for the event. Event names can be alphanumeric, can contain underlines, are case-sensitive, and must start with a letter. An event cannot have the same name as another event, term, or qualifier.

3) In the Range field, right-click to select a signal output for the event. For more information about the range field, see *Defining Range Logic Output* on page 4-27.

4) In the comments field, enter any comments you want to include about the event.

5) Click on OK. The BTT application checks the event for validity. If you have created a valid event, the BTT application displays the event in the Events window.

4.7 Building a Trace Language Program

After you have created the terms and events, your next step is to build them into a trace language program that the range logic on the XDS512RL board can understand.

To build a trace language program, press on the toolbar or select Build from the Program menu. When you build your trace language program, the BTT application verifies that there are no errors or conflicts in your event and term definitions. If you have errors in your event or term definitions, the BTT application displays a list of errors. You can then double-click on an individual error to display and correct the erroneous definition.

4.8 Programming the Range Logic

To program the range logic with a trace language program, press on the toolbar, or select Program from the Program menu. The program command programs the range logic with the current trace language program displayed in the interface.

When you use the program command, the BTT application builds/verifies the trace language program before programming it into the range logic. If the BTT application detects errors in the trace language program or if it cannot initialize the XDS512RL, then the program button and menu selection are grayed out.

4.9 Initializing the XDS512RL

When you invoke the BTT application, it tries to initialize the XDS512RL. If the BTT application cannot initialize the XDS512RL, it displays an error message that tells you what the problem is. After correcting any problems, you can reinitialize the XDS512 by selecting *Initialize XDS512* from the *Program* menu.

If the BTT application cannot initialize the XDS512RL, you cannot program the range logic and the program commands on the menubar and toolbar are grayed out.

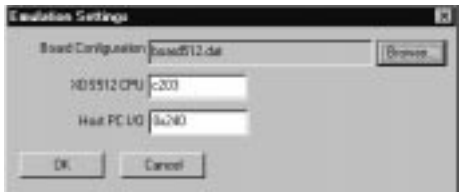
4.10 Emulation Settings

You use the Emulation Settings window to tell the BTT application where your CPUs are on the XDS512RL board. This is necessary because the board contains two processors, a TMS320C2xx that is part of the XDS512RL board, and your TMS320C27x. To define your emulation settings, you must provide the following information:

Table 4–12. Emulation Settings

Information	Description
Board Configuration	Enter the name of the configuration file, such as *board.dat*, that contains your emulation settings data for the TMS320C27x source code debugger. If you cannot remember the name of the file, click Set to display the Configuration File Selection dialog box.
XDS512 CPU	Enter the name of the processor you are using. The name corresponds to the name of the processor in your board.dat file.
Host PC I/O:	Enter, in hexadecimal, the PC I/O address of the XDS510 controller card.

Table 4–13. The Emulation Settings Window



4.10.1 Defining Your Emulation Settings

To define your emulation settings, follow these steps:

1) From the Program menu, select Emulation Settings... to display the Emulation Settings dialog box.

2) In the Board Configuration field, enter the name of your configuration file. If you cannot remember the name of your board configuration file, click Browse... to display the Configuration File Selection dialog box.

3) Enter the name of the CPU that you want to configure.

4) Enter, in hexadecimal, the PC I/O address of the XDS510 controller card.

Appendix A

Trace Language Grammar

Use the trace language to program the range logic on the XDS512RL. When you use the BTT application, the interface translates your input into a trace language program (file.btt). This chapter defines and explains the grammar and syntax of the trace language.

Topic **Page**

Notational Conventions

This appendix uses the following conventions:

- In syntax descriptions, portions of syntax that are in **bold** should be entered as shown; portions of syntax that are in *italics* describe the type of information that should be entered.
- Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you must specify the information within the brackets. Unless the brackets are in a **bold** typeface, do not enter the brackets themselves. Here is an example of these conventions:

Term *TermName* **{** [*CommentBlock*] *TermExpression* **}**

Term is a portion of the grammar that you must enter as shown and is followed by the *TermName* parameter for which you must provide a value. The opening brace, **{**, is required by the syntax and is followed by the optional parameter [*CommentBlock*]. Finally, you must provide a value for the *TermExpression* parameter and end the statement with a closing brace, **}**.

A.1 Event Definition Statement

The syntax for an event definition statement is as follows:

Event *EventName Range* **{** [*//CommentBlock*] *EventExpression* **}**

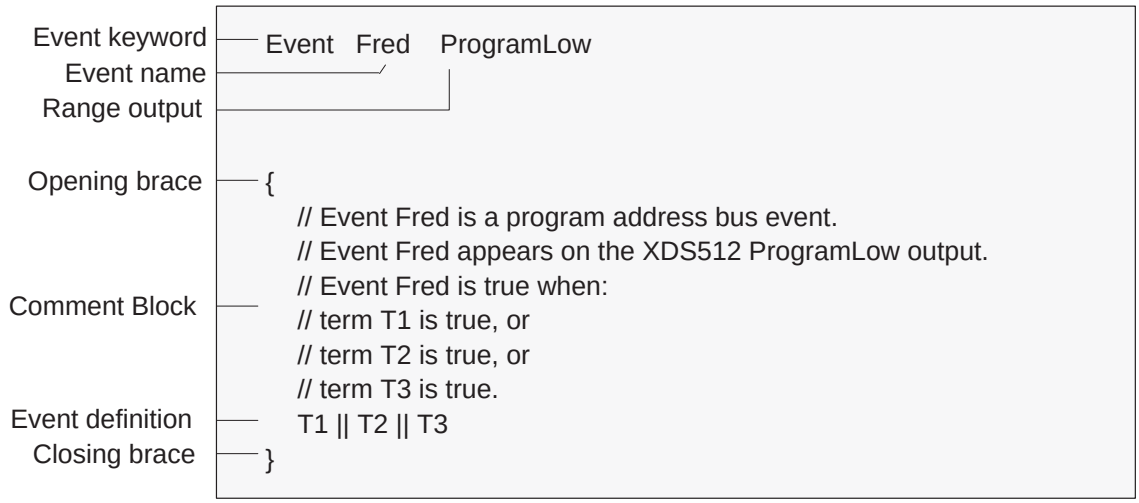
- The **Event** keyword begins each event definition. It is case sensitive, so you must enter it as shown.
- ***EventName*** assigns a name to the event you are defining. It is case sensitive and must start with a letter. It can be alphanumeric; however, it cannot contain a space or punctuation. An event cannot have the same name as a term.
- ***Range*** indicates which XDS512RL output you want the event to use. It can have one of the following values:
  - ProgramLow, Program4, Program5, Program6, Program7
  - ReadLow, ReadHigh, Read4, Read5, Read6, Read7
  - WriteLow, WriteHigh, Write4, Write5, Write6, Write7

  These values are case-sensitive and must be entered as shown.
- ***//CommentBlock*** contains any comments you want to include about the event you are defining. Comments are allowed only at the beginning of a trace language file or just within the opening brace of a term or event definition. Each comment line must start with //. Comments cannot share a line with other code; they must be on a separate line.
- The ***EventExpression*** parameter is one or more terms ORed together. For example, given three terms named T1, T2, and T3, an event expression could be:

  T1 || T2 || T3

Example A–1 shows trace language that defines an event.

Example A–1. Defining an Event With the Trace Language



A.2 Term Definition Statement

The syntax for a trace definition statement is as follows:

Term *TermName* **{** [*//CommentBlock*] *TermExpression* **}**

- The **Term** keyword begins each term definition. It is case sensitive, so you must enter it as shown.
- ***TermName*** assigns a name to the term you are defining. It is case sensitive and must start with a letter. It can be alphanumeric; however, it cannot contain a space or punctuation. A term cannot have the same name as an event.
- ***//CommentBlock*** contains any comments you want to include about the term you are defining. Comments are allowed only at the beginning of a trace language file or just within the opening brace of a term or event definition. Each comment line must start with //. Comments cannot share a line with other code; they must be on a separate line.
- The ***TermExpression*** parameter defines a value for the term. This parameter can be an address expression, a qualifier expression, or an address expression logically ANDed with a qualifier expression. Sections A.2.1 and A.2.2 describe how to write address expressions and qualifier expressions, respectively.

Section A.2.3 gives examples of terms defined with the trace language.

A.2.1 Writing Address Expressions

An address expression consists of one or more relational expressions or range expressions ORed together.

- Relational expressions use the == operator to compare a bus's current value to a constant value. The syntax for defining a relational expression is as follows:

  [**(**] *BusName* **= =** *AddressSpecifier* [**)**]

- Range expressions use the [ ] operator to determine if the bus's current value is in the specified range. Ranges are inclusive. The syntax for defining a range expression is as follows:

  [**(**] *BusName* **[ ]** *RangeSpecifier* [**)**]

The values for the *BusName*, *AddressSpecifier*, and *RangeSpecifier* parameters shown above are defined in the following table. Table 4–2 provides examples of address expressions.

Parameter	Choices	Description
BusName	PA	Program address bus
	RA	Data-read address bus
	WA	Data-write address bus
AddressSpecifier	Constant	A hexadecimal or decimal number.
	Symbol +/– Constant	Symbol is a symbol that has been assigned to an address. Constant, which is optional, is a hexadecimal or decimal offset.
RangeSpecifier	(Start Address, End Address)	For Start Address and End Address, you can use numeric addresses (hexadecimal or decimal) or a symbolic address (for example,_main + 2).
	(Start Address,n**L**)	For Start Address, you can use numeric addresses (hexadecimal or decimal) or a symbolic address (for example,_main + 2). For n, you specify a hexadecimal or decimal constant that specifies the length of the address range (including the start address). The L is required to indicate that you are specifying a length instead of an end address.

Table A–1. Examples of Address Expressions

Example Address Expression	Description
WA==0x200 || WA[] (0x300,0x400)	This expression is true when either of the following appears on the data-write address bus: ❑ The address 0x200. ❑ An address in the range from 0x300 to 0x400.
RA[] (0x100,0x200) || RA[] (0x300,10L)	This expression is true when either of the following appears on the data-read address bus: ❑ An address in the range from 0x100 to 0x200. ❑ An address in the range from 0x300 to 0x309.
PA[] (_asmfunc1,_asmfunc2 – 1)	This expression uses the symbols _asmfunc1 and _asmfunc2 to represent two program-space addresses. The expression is true when the program address bus carries an address in the range from _asmfunc1 to (_asmfunc2 – 1).

A.2.2 Writing Qualifier Expressions

Qualifiers allow you to make your terms partly or solely dependent on particular conditions in the CPU. Qualifier expressions are built up from qualifier names and the operators in the following table. The && operator has precedence over the || operator. When you need to override precedence, you can enclose subexpressions in parentheses; these subexpressions will be evaluated first.

Operator	Description
&&	AND
||	OR
(	Opening parenthesis
)	Closing parenthesis

Table A–2 shows examples of qualifier expressions. Qualifiers are described in detail in section 4.5 on page 4-21.

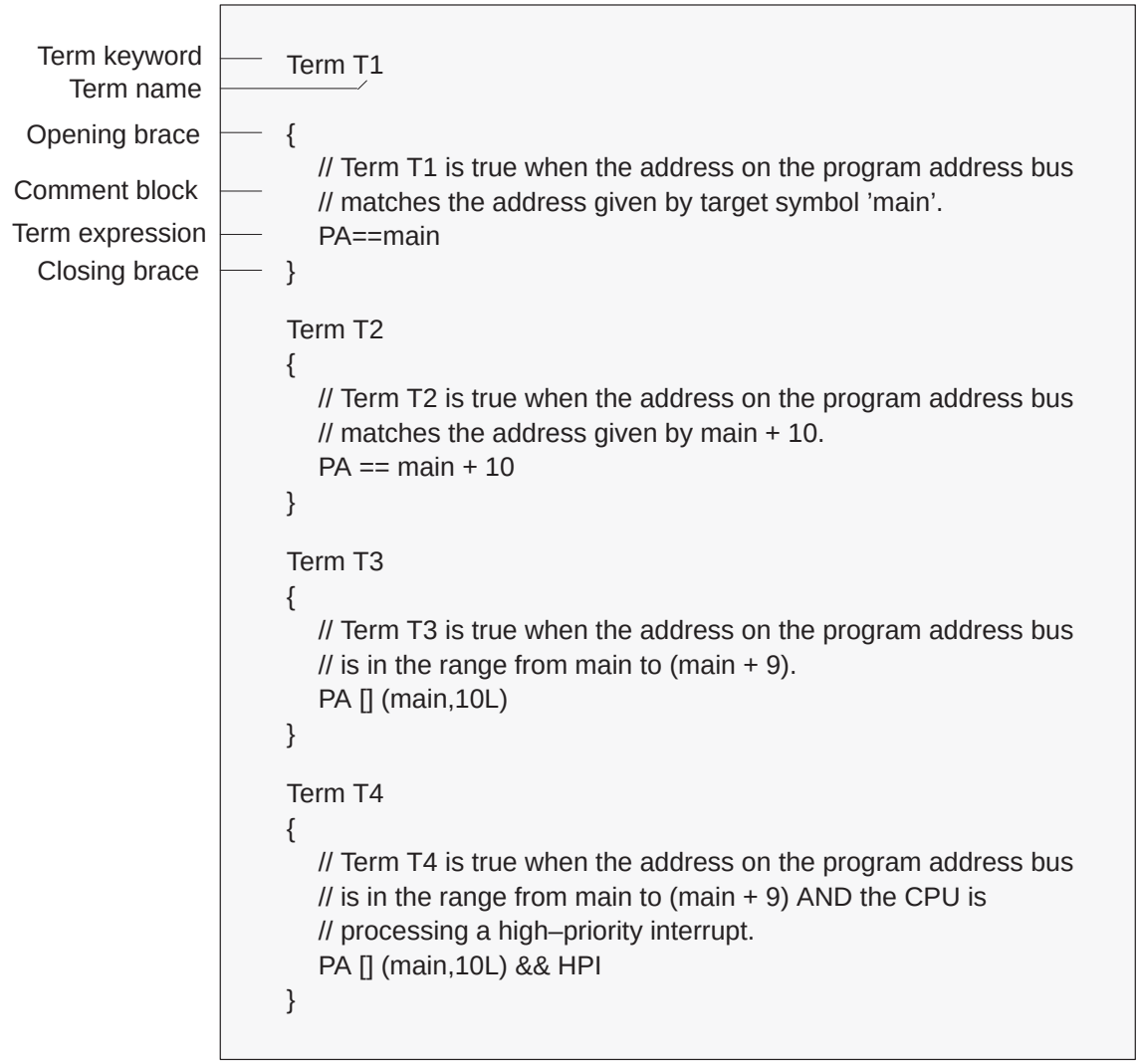
Table A–2. Examples of Qualifier Expressions

Example Qualifier Expression	Description
HPI	This expression is true if the CPU is processing a high-priority interrupt (HPI).
HPI || Last	This expression is true if the CPU is processing a high-priority interrupt OR the CPU is executing the last instruction before a discontinuity.
NotHPI && TrueBranch	This expression is true if the CPU is not processing a high-priority interrupt AND the CPU has tested a branch condition and found it to be true.
(First || Last) && NotHPI	This expression is true if *both* of the following conditions are true: ❑ The CPU is executing the first instruction after a discontinuity OR the CPU is executing the last instruction before a discontinuity. ❑ The CPU is not processing a high-priority interrupt.
PageZeroDirect && Any16BitRead	This expression is true if PAGE0 direct addressing mode is selected AND the CPU is reading a 16-bit value from data space.
(AnyLowByteStackWrite && HPI) || (AnyHighByteStackWrite && HPI)	This expression is true if *either* of the following conditions is true: ❑ The CPU is writing to the low byte of a stack location AND the CPU is processing a high-priority interrupt. ❑ The CPU is writing to the high byte of a stack location AND the CPU is processing a high-priority interrupt.

A.2.3 Examples of Term Definition Statements

Example A–2 shows trace language code that defines six terms, T1–T6. Terms T1–T3 use only an address expression. Term T5 uses only a qualifier expression. Terms T4 and T6 each combine an address expression and a qualifier expression.

Example A–2. Defining Terms With the Trace Language



Example A–2. Defining Terms With the Trace Language (Continued)

```
Term T5
{
    // Term T5 is true when the CPU writes a 16–bit value to a nonstack
    // location at an odd address.
    Odd16BitNonStackWrite
}

Term T6
{
    // Term T6 is true when:
    // The address on the program address bus in is the range
    //     from 0x0 to 0x100
    // AND the instruction being executed is the first one after a
    //     PC discontinuity or the last one before a PC discontinuity
    // AND the CPU is processing a high–priority interrupt.
    PA [] (0,0x100) && (First || Last) && HPI
}
```

A.3 Grouping Statement

With a Grouping statement, you can set the grouping status for each address bus (program, read, and write). When grouping is on for a bus, every corresponding signal connected to the TLA704 is asserted for all events associated with that bus (see section 2.3, *Grouping Range Logic Output Signals*, on page 2-7).

The syntax for a Grouping statement is as follows:

Grouping *BusSetting1 [BusSetting2] [BusSetting3]*

- The **Grouping** keyword begins each Grouping statement. It is case sensitive, so you must enter it as shown.
- ***BusSetting*** assigns an on/off value to the grouping setting of a particular bus. You can have up to three *BusSetting* parameters, one for each bus (program, read, or write). The possible *BusSetting values* are listed in the following table. These keywords are case sensitive and must be entered as shown.

Bus	***BusSetting* Choices**
Program	ProgramOn or ProgramOff
Read	ReadOn or ReadOff
Write	WriteOn or WriteOff

You use the Grouping statement once in your file to set the grouping status for all events in that file. If you do not specify an On or Off parameter for a given bus, the default for each bus is Grouping Off. Table A–3 shows examples of grouping statements.

Table A–3. Examples of Grouping Statements

Example Grouping Statement	Results
Grouping ProgramOn ReadOff	Program bus: Grouping On Read bus: Grouping Off Write bus: Grouping Off (default)
Grouping ReadOn	Program bus: Grouping Off (default) Read bus: Grouping On Write bus: Grouping Off (default)
Grouping ProgramOn ReadOn WriteOn	Program bus: Grouping On Read bus: Grouping On Write bus: Grouping On

A.4 Example Trace Language Program

Trace language files have the extension .btt. Example A–3 illustrates how to use the trace language to set grouping status, create terms, and create events.

Example A–3. Trace Language Program

```
// Program comment
// Another program comment

// Load the target executable (.out) file that the BTT application will
// use to find the definition of target symbols used in a terms' address
// expressions.
#load "C:\MyDir\Target.out"

// The Grouping statement below turns on grouping for the program and
// read buses. Because the write bus is not specified, grouping is off for
// the write bus by default.
Grouping ProgramOn ReadOn

Term T1
{
   // Term T1 is true when the address on the program address bus
   // matches the address given by target symbol 'main'.
   PA == main
}

Term T2
{
   // Term T2 is true when the program address bus carries:
   // An address in the range from 0x500 to 0x502
   // OR the address given by the target symbol Function2.
   PA [] (0x500,3L) || PA == Function2
}

Term T3
{
   // Term T3 is true when conditional branch instruction is executed and
   // the condition is false.
   FalseBranch
}
```

Example A–3. Trace Language Program (Continued)

```
Event Fred Program4
{
   // Event Fred is a program address bus event.
   // Event Fred appears on XDS512RL program trigger 4.
   // Event Fred is true when:
   // Term T1 is true
   // OR Term T2 is true.
   T1 || T2
}

Event Barney ProgramLow
{
   // Event Barney triggers the ProgramLow signal.
   // Event Barney is true when Term T3 is true.
   T3
}

// These are illegal comments. Comments are allowed only at the
// beginning of a file or just within the opening brace of a Term or
// Event definition statement.
//
// The following Term and Event definition statements cause errors
// for the reasons explained in their comment blocks.

Term T2
{
   // Term T2 has already been defined. This is an illegal redefinition
   // of Term T2
   PA [] (0x500,3L) || PA == Function2
}

Term Fred
{
   // This is an illegal term definition because the name Fred has
   // already been assigned to Event Fred. Events and terms cannot
   // have the same names.
   PA == 0x400
}
```

Example A–3. Trace Language Program (Continued)

```
Event Wilma
{
    // This event definition is illegal because no range logic output signal
    // has been specified after the name Wilma.
    T2
}

Event Pebbles WriteLow
{
    // This is an illegal event definition because it does not contain a
    // proper event expression. The expression must be one or more
    // term names separated by the OR operator, ||. In addition, the
    // terms named in the expression must be defined in the file. To
    // correct this error, a programmer would define a term that contains
    // the address expression WA == 0x102 and then use the term name
    // in this event definition.
    WA  ==  0x102
}

Event BamBam Read5
{
    // This event definition is illegal because it has a conflict between
    // the range output signal and the bus monitored by Term T1. Event
    // BamBam is set to trigger a read signal, but Term T1 has been set
    // to monitor the program address bus (PA == main).
    T1
}
```

Appendix B

Qualifier Bit Masks

B.1 Qualifier Bit Masks

This appendix shows bit masks that are written to the SRAMs for each qualifier. In the following tables, X indicates a don't care; the value may be 0 or 1.

Table B–1. Program Qualifiers and Their Corresponding Bit Masks

	Bit							
	14	**13**	**12**	**11**	**10**	**9**	**8**	**7**
Program Qualifier	**(GND)**	**LST**	**FST**	**PATH(1)**	**PATH(0)**	**HPI**	**INTJAM**	**INSTR**
HPI	X	X	X	X	X	1	X	X
NotHPI	X	X	X	X	X	0	X	X
Execute	X	X	X	X	X	X	X	1
IntJam	X	X	X	X	X	X	1	1
NotIntJam	X	X	X	X	X	X	0	1
First	X	X	1	X	X	X	X	1
Last	X	1	X	X	X	X	X	1
Middle	X	0	0	X	X	X	X	1
IsBranch† 1st part	X	X	X	0	1	X	X	1
2nd part	X	X	X	1	0	X	X	1
IsNotBranch	X	X	X	0	0	X	X	1
TrueBranch	X	X	X	0	1	X	X	1
FalseBranch	X	X	X	1	0	X	X	1
IndirectOrCall	X	X	X	1	1	X	X	1
PipeStall	X	X	X	X	X	X	X	0

† This qualifer requires two writes to the SRAM.

Table B–2. Data-Read Qualifiers and Their Corresponding Bit Masks

		Bit							
		14	13	12	11	10	9	8	7
Data-Read Qualifier		**PAGE0**	**HPI**	**RBU(2)**	**RBU(1)**	**RBU(0)**	**RPM**	**SPR**	**RDM**
HPI		X	1	X	X	X	X	X	X
NotHPI		X	0	X	X	X	X	X	X
PageZeroStack		1	X	X	X	X	X	X	X
PageZeroDirect		0	X	X	X	X	X	X	X
AnyRead		X	X	X	X	X	X	X	X
NoAccessRead		X	X	X	X	X	0	0	0
AnyProgramRead		X	X	X	X	X	1	0	0
AnyStackRead		X	X	X	X	X	0	1	1
AnyNonStackRead		X	X	X	X	X	0	0	1
Any16BitProgramRead†	1st part	X	X	0	1	0	1	0	0
	2nd part	X	X	0	0	1	1	0	0
Even16BitProgramRead		X	X	0	0	1	1	0	0
Odd16BitProgramRead		X	X	0	1	0	1	0	0
A32BitStackRead		X	X	0	1	1	0	1	1
Any16BitStackRead†	1st part	X	X	0	0	1	0	1	1
	2nd part	X	X	0	1	0	0	1	1
Even16BitStackRead		X	X	0	0	1	0	1	1
Odd16BitStackRead		X	X	0	1	0	0	1	1
AnyLowByteStackRead†	1st part	X	X	1	0	0	0	1	1
	2nd part	X	X	1	1	0	0	1	1
EvenLowByteStackRead		X	X	1	0	0	0	1	1
OddLowByteStackRead		X	X	1	1	0	0	1	1

† This qualifer requires two writes to the SRAM.

Table B–2. Data-Read Qualifiers and Their Corresponding Bit Masks (Continued)

		Bit							
		14	13	12	11	10	9	8	7
Data-Read Qualifier		**PAGE0**	**HPI**	**RBU(2)**	**RBU(1)**	**RBU(0)**	**RPM**	**SPR**	**RDM**
AnyHighByteStackRead†	1st part	X	X	1	0	1	0	1	1
	2nd part	X	X	1	1	1	0	1	1
EvenHighByteStackRead		X	X	1	0	1	0	1	1
OddHighByteStackRead		X	X	1	1	1	0	1	1
Any16BitNonStackRead†	1st part	X	X	0	0	1	0	0	1
	2nd part	X	X	0	1	0	0	0	1
Even16BitNonStackRead		X	X	0	0	1	0	0	1
Odd16BitNonStackRead		X	X	0	1	0	0	0	1
A32BitNonStackRead		X	X	0	1	1	0	0	1
AnyLowByteNonStackRead†	1st part	X	X	1	0	0	0	0	1
	2nd part	X	X	1	1	0	0	0	1
EvenLowByteNonStackRead		X	X	1	0	0	0	0	1
OddLowByteNonStackRead		X	X	1	1	0	0	0	1
AnyHighByteNonStackRead†	1st part	X	X	1	0	1	0	0	1
	2nd part	X	X	1	1	1	0	0	1
EvenHighByteNonStackRead		X	X	1	0	1	0	0	1
OddHighByteNonStackRead		X	X	1	1	1	0	0	1

† This qualifer requires two writes to the SRAM.

Table B–3. Data-Write Qualifiers and Their Corresponding Bit Masks

	Bit							
	14	13	12	11	10	9	8	7
Data-Read Qualifier	**PAGE0**	**HPI**	**WBU(2)**	**WBU(1)**	**WBU(0)**	**WPM**	**SPW**	**WDM**
HPI	X	1	X	X	X	X	X	X
NotHPI	X	0	X	X	X	X	X	X

Table B–3. Data-Write Qualifiers and Their Corresponding Bit Masks (Continued)

		Bit							
		14	13	12	11	10	9	8	7
Data-Read Qualifier		**PAGE0**	**HPI**	**WBU(2)**	**WBU(1)**	**WBU(0)**	**WPM**	**SPW**	**WDM**
PageZeroStack		1	X	X	X	X	X	X	X
PageZeroDirect		0	X	X	X	X	X	X	X
AnyWrite		X	X	X	X	X	X	X	X
NoAccessWrite		X	X	X	X	X	0	0	0
AnyProgramWrite		X	X	X	X	X	1	0	0
AnyStackWrite		X	X	X	X	X	0	1	1
AnyNonStackWrite		X	X	X	X	X	0	0	1
Any16BitProgramWrite†	1st part	X	X	0	1	0	1	0	0
	2nd part	X	X	0	0	1	1	0	0
Even16BitProgramWrite		X	X	0	0	1	1	0	0
Odd16BitProgramWrite		X	X	0	1	0	1	0	0
A32BitStackWrite		X	X	0	1	1	0	1	1
Any16BitStackWrite†	1st part	X	X	0	0	1	0	1	1
	2nd part	X	X	0	1	0	0	1	1
Even16BitStackWrite		X	X	0	0	1	0	1	1
Odd16BitStackWrite		X	X	0	1	0	0	1	1
AnyLowByteStackWrite†	1st part	X	X	1	0	0	0	1	1
	2nd part	X	X	1	1	0	0	1	1
EvenLowByteStackWrite		X	X	1	0	0	0	1	1
OddLowByteStackWrite		X	X	1	1	0	0	1	1
AnyHighByteStackWrite†	1st part	X	X	1	0	1	0	1	1
	2nd part	X	X	1	1	1	0	1	1
EvenHighByteStackWrite		X	X	1	0	1	0	1	1

† This qualifer requires two writes to the SRAM.

Table B–3. Data-Write Qualifiers and Their Corresponding Bit Masks (Continued)

		Bit							
		14	13	12	11	10	9	8	7
Data-Read Qualifier		**PAGE0**	**HPI**	**WBU(2)**	**WBU(1)**	**WBU(0)**	**WPM**	**SPW**	**WDM**
OddHighByteStackWrite		X	X	1	1	1	0	1	1
Any16BitNonStackWrite†	1st part	X	X	0	0	1	0	0	1
	2nd part	X	X	0	1	0	0	0	1
Even16BitNonStackWrite		X	X	0	0	1	0	0	1
Odd16BitNonStackWrite		X	X	0	1	0	0	0	1
A32BitNonStackWrite		X	X	0	1	1	0	0	1
AnyLowByteNonStackWrite†	1st part	X	X	1	0	0	0	0	1
	2nd part	X	X	1	1	0	0	0	1
EvenLowByteNonStackWrite		X	X	1	0	0	0	0	1
OddLowByteNonStackWrite		X	X	1	1	0	0	0	1
AnyHighByteNonStackWrite†	1st part	X	X	1	0	1	0	0	1
	2nd part	X	X	1	1	1	0	0	1
EvenHighByteNonStackWrite		X	X	1	0	1	0	0	1
OddHighByteNonStackWrite		X	X	1	1	1	0	0	1

† This qualifer requires two writes to the SRAM.

Appendix C

Configuring the XDS512RL

This appendix explains how to configure your XDS512RL for proper operation of your emulation system.

Topic **Page**

C.1 XDS512RL Diagram

Figure C–1 shows the locations of connectors, jumper groups, and switches and other items on the emulator board. Table C–1 contains descriptions of these items.

Figure C–1. Layout of the XDS512RL

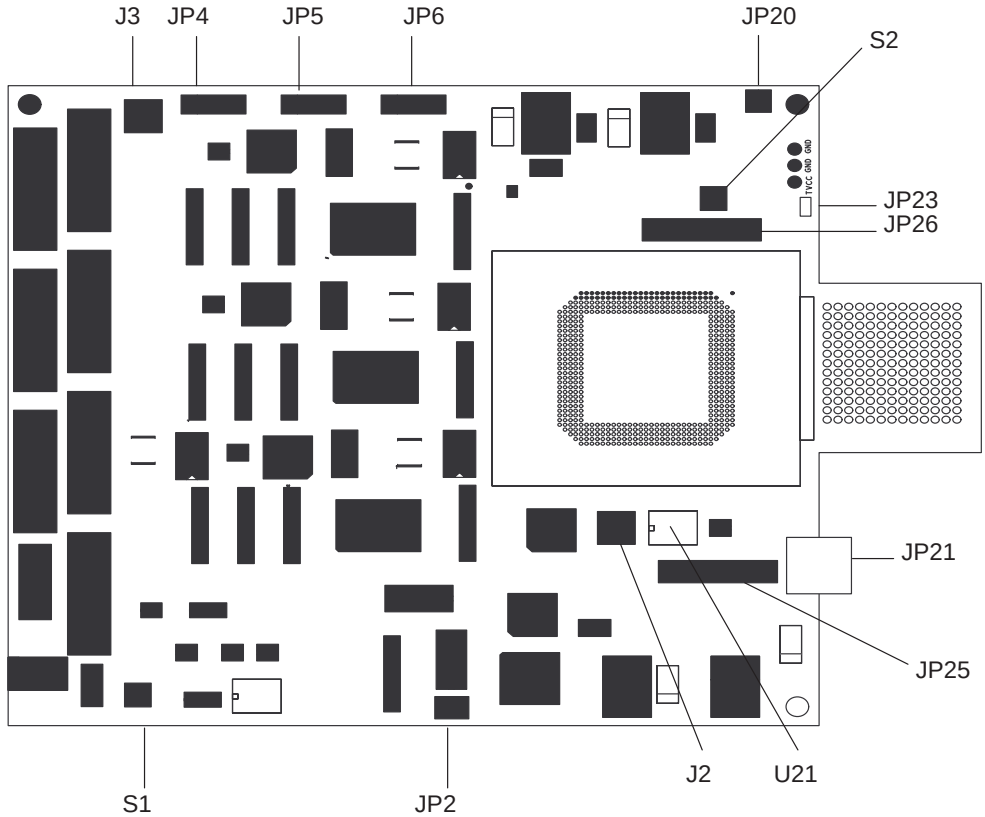


Table C–1. Connectors, Jumper Groups, Switches, Etc., on the XDS512RL

Item	Description	Section(s) To See ...
J2	*Oscillator input connector.* Use this connector to connect an external clock source to the XDS512RL.	Using an external SMA clock source, page C-13
J3	This connector is reserved for test purposes. Do not use this connector.	
JP2	*Jumper group 2: Operating mode.* Use this jumper group to choose a power-on operating mode for the XDS512RL.	Setting an operating mode, page C-5

Table C–1. Connectors, Jumper Groups, Switches, Etc., on the XDS512RL (Continued)

Item	Description	Section(s) To See ...
JP4	*Jumper group 4: Program-space range-logic signals.* Use JP4 to access signals generated by the program-space range logic on the XDS512RL.	Accessing range-logic signals at XDS512RL pins, page C-24
JP5	*Jumper group 5: Read-space range-logic signals.* Use JP5 to access signals generated by the read-space range logic on the XDS512RL.	Accessing range-logic signals at XDS512RL pins, page C-24
JP6	*Jumper group 6: Write-space range-logic signals.* Use JP6 to access signals generated by the write-space range logic on the XDS512RL.	Accessing range-logic signals at XDS512RL pins, page C-24
JP20	*Jumper group 20: Target Reset Options.* Use JP20 to set target reset options.	Setting target reset options, page C-7
JP21	*Jumper group 21: Probes.* Use JP21 to pass signals from the target device to the TLA704.	Routing target signals to the TLA704, page C-22
JP23	*Jumper group 23: Power mode.* Use JP23 to isolate the XDS512RL and target power sources or to have the target share the XDS512RL power source.	Setting a power mode, page C-9
JP25	*Jumper group 25: Clock configuration.* Use JP25 to control the source and frequency of the XDS512RL functional clock.	Selecting a clock source, page C-12 Modifying the clock frequency, page C-14 Complete pin listings for JP25 and JP26, page C-19
JP26	*Jumper group 26: Control.* Use JP26 to enable or disable the phase-lock loop (PLL) and to choose one of two locations for the interrupt vectors.	Multiplying the clock frequency with the PLL, page C-15 Mapping the TMX320C2700–E1 interrupt vectors, page C-10 Complete pin listings for JP25 and JP26, page C-19
S1	*XDS512RL reset switch.* Press this switch to reset the target system.	

Table C–1. Connectors, Jumper Groups, Switches, Etc., on the XDS512RL (Continued)

Item	Description	Section(s) To See ...
S2	*Target reset switch.* Press this switch to reset the XDS512RL.	Setting target reset options, page C-7
U21	*XDS512RL oscillator.* This 10-MHz oscillator can be chosen as the clock source for the TMX320C2700–E1.	Selecting a clock source, page C-12 Removing the XDS512RL oscillator, page C-21 Reinstalling the XDS512RL oscillator, page C-21

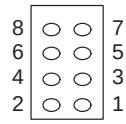
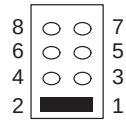
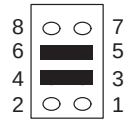
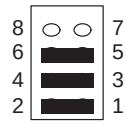
C.2 Setting an Operating Mode

The XDS512RL operating mode is determined by jumper group JP2, as shown in Table C–2. If you are using an XDS510 emulator, use one of the *emulator* modes. If no XDS510 emulator is connected to your XDS512RL, use one of the *stand-alone* modes. The JP2 setting is read at power-up. Each time you wish to use a new operating mode, you must remove power from the XDS512RL and then power it up again.

The column in Table C–2 labeled MODE(2:0) refers to the three signals that determine the operating mode (see the JP2 pin descriptions in Table C–3). If a MODE signal is jumpered with its corresponding ground signal, it represents a logic 0. If a MODE signal is not jumpered, it represents a logic 1. For example, adding a jumper across pins 1 and 2 gives you MODE0 = 0; removing the jumper gives you MODE0 = 1.

Do not place a jumper across JP2 pins 7 and 8. Connecting pins 7 and 8 can damage your XDS512RL.

Table C–2. Setting an XDS512RL Operating Mode With JP2

XDS510 Emulator Connected to XDS512RL?	Operating Mode	JP2 Setting†	MODE(2:0)
Yes	*Emulator hold-in-reset.* The emulation chip is held in reset after power-up. You can use this mode to hold the emulation chip in reset while you download your code. This mode is for operation with an XDS510 emulator.		111
Yes	*Emulator run-from-reset.* The emulation chip runs after power-up and reset. This mode is for operation with an XDS510 emulator.	 **Default setting**	110
No	*Stand-alone hold-in-reset.* The emulation chip is held in reset after power-up. You can use this mode to hold the emulation chip in reset mode while you download your code. This mode is for operation without an XDS510 emulator.		001
No	*Stand-alone run-from-reset.* The emulation chip runs after power-up and reset. This mode is for operation without an XDS510 emulator.		000

† Other JP2 settings are reserved and should not be used. Only these four modes are available.

Table C–3. JP2 Pins

Pin	Signal	Description
1	MODE0	Least significant operating mode bit
2	GND	Ground
3	MODE1	Middle operating mode bit
4	GND	Ground
5	MODE2	Most significant operating mode bit
6	GND	Ground
7	Reserved	Do not use this pin.
8	Reserved	Do not use this pin.

C.3 Setting Target Reset Options

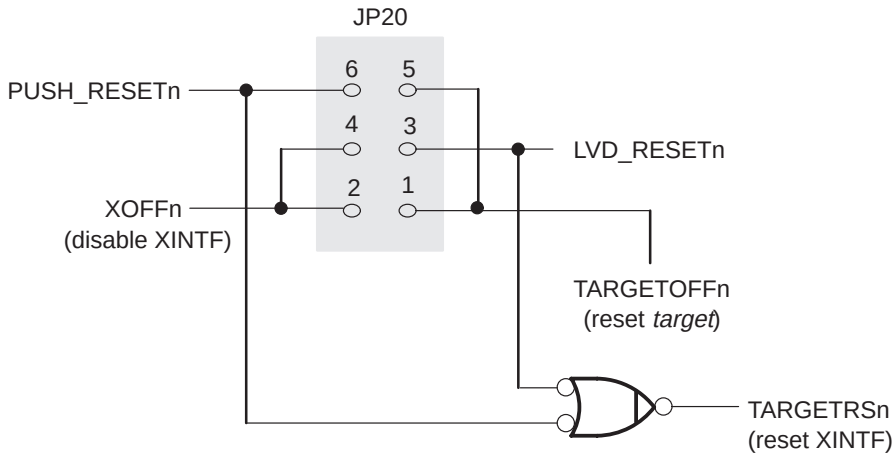
Target reset options are determined by jumper group JP20, which has the pins described in Table C–4. You may jumper one or more of the pin sets 1–2, 3–4, and 5–6 to change XDS512RL reset operations. These operations effect the target system and the external interface (XINTF) that enables communication between the emulation chip's core and the target system.

Figure C–2 shows the positions of the JP20 signals. Regardless of the JP20 setting, TARGETRSn is asserted in response to either a manual reset (PUSH_RESETn active) or a low-voltage condition in the target system (LVD_RESETn active). Adding jumpers to JP20 changes other signal relationships. Table C–5 explains the new signal relationships that arise for each jumpered pin set. As you read the table, you may want to study the signal paths in Figure C–2. Each signal is driven by an open-collector driver.

Table C–4. JP20 Pins

Pin	Signal	Description
1	TARGETOFFn	This is a bidirectional signal. It can be used as an input to the target to tell the target to reset. It can also be used as an output from the target to tell the emulation chip that the target is resetting.
2	XOFFn	This is an input to the emulation chip. When low, XOFFn disables the external interface (XINTF) used for communication between the emulation chip and the target.
3	LVD_RESETn	*Low-voltage-detector reset.* When low, LVD_RESETn indicates that the supply voltage of the target system has dropped below the minimum required to run the target system.
4	XOFFn	This is the same signal that is on pin 2.
5	TARGETOFFn	This is the same signal that is on pin 1.
6	PUSH_RESETn	*Push-button reset.* This signal is connected to the push button at the position labeled S2 on the XDS512RL. When you press this button, PUSH_RESETn is active. If you jumper pins 5 and 6, you can use this button to manually force a target reset condition.

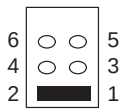
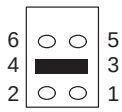
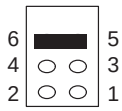
Figure C–2. JP20 Diagram



Note:

Regardless of the JP20 setting, TARGETRSn is asserted in response to either a manual reset (PUSH_RESETn active) or a low-voltage condition (LVD_RESETn active).

Table C–5. Effects of Jumpering Each JP20 Pin Set

JP20 Setting	New Signal Relationship	Result
 **Default setting**	XOFFn and TARGETOFFn are asserted and released together.	XINTF is disabled whenever the target is reset, and the target is reset whenever XINTF is disabled.
	If LVD_RESETn asserted XOFFn asserted	A low-voltage condition disables XINTF (in addition to asserting TARGETRSn).
	If PUSH_RESETn asserted TARGETOFFn asserted	If you press the target reset button (S2), you manually reset the target (in addition to asserting TARGETRSn).

C.4 Setting a Power Mode

The following table shows the two power modes you can select with JP23. If you want the target system to use the same power source as the XDS512RL, make sure there is a jumper across JP23. If you want to use an independent power source for the target system, make sure JP23 has no jumper. In either mode, the target and the XDS512RL share the same ground. Table C–7 describes the two pins of JP23.

Table C–6. Setting an XDS512RL Power Mode With JP23

Power Mode	JP23 Setting
Power shared. The target system and the XDS512RL share the XDS512RL power source.	 **Default setting**
Power isolated. The target system and the XDS512RL use independent power sources.	

Table C–7. JP23 Pins

Pins	Signal	Description
1	TVCC	Supply voltage for the target system
2	XDS_5V	Supply voltage for the XDS512RL

C.5 Mapping the TMX320C2700–E1 Interrupt Vectors

The TMX320C2700–E1 interrupt vectors can be mapped to one of two program-address ranges using pins 13 and 14 of jumper group JP26. Table C–8 describes the options and how to choose them with pins 13 and 14. The status of the other pins is not shown because they do not affect the mapping of the interrupt vectors. Table C–9 defines pins 13 and 14. For more information about JP26, see section C.6.4 on page C-19.

Table C–8. Mapping the TMX320C2700–E1 Interrupt Vectors With JP26:13–14

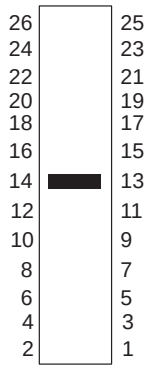
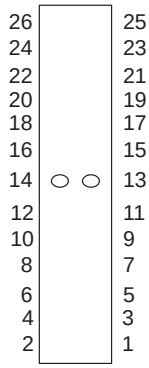
Location of TMX320C2700–E1 Interrupt Vectors	JP26:13–14 Setting
Interrupt vectors mapped to program-space addresses 0x000000–0x00003F. When the TMX320C2700–E1 is reset, the reset vector is fetched at address 0x000000.	
Interrupt vectors mapped to program-space addresses 0x3FFFC0–0x3FFFFF. When the TMX320C2700–E1 is reset, the reset vector is fetched at address 0x3FFFC0.	

Table C–9. JP26 Pins 13 and 14

Pin(s)	Signal	Description
13	VMAP	Interrupt vector map signal
14	GND	Ground

C.6 Setting Up a Clock for the TMX320C2700–E1

This section explains how to select an XDS512RL clock source and modify its frequency. The following table lists the topics covered:

C.6.1 Selecting A Clock Source

The clock source for the emulation chip can be a target signal or a non-target signal. You have two choices for a non-target signal: a 10-MHz signal generated by the XDS512RL oscillator or an external signal connected to the XDS512RL.

JP25 pins 1 and 2 (JP25:1–2) determine whether the XDS512RL uses a target or non-target clock source for the emulation chip. Table C–10 shows how to select either option. Table C–11 identifies JP25 pins 1 and 2. (For a complete pin description for JP25, see section C.6.4 on page C-19.)

Table C–10. Selecting a Target or Non-Target Clock Source With JP25:1–2

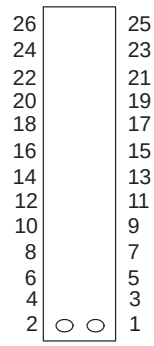
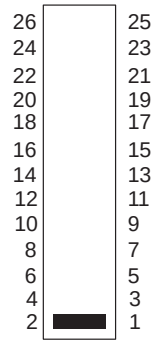
Target or Non-Target Clock Source?	JP25:1–2 Setting
Target clock source. The target provides a source clock by way of the pin grid array (see section C.6.1.1 on page C-13).	
Non-target clock source. You can use either of the following for a clock source: ❑ The on-board oscillator (10 MHz). For more details about using the oscillator see section C.6.1.2 on page C-13. ❑ An external clock signal. For more details about using an external clock source, see section C.6.1.3 on page C-13.	

Table C–11. JP25 Pins 1 and 2

Pin(s)	Signal	Description
1	CLKSELn	Is used to select a target or non-target clock source
2	GND	Ground

C.6.1.1 Using A Target System Clock

To use a target system clock as a clock source, locate jumper group JP25 and make sure pins 1 and 2 are *not* jumpered. This indicates that the clock source is taken from pin K7 (TARGETCLK) of the pin grid array (PGA) that connects the target system to the XDS512RL.

To modify the frequency of the clock signal before it reaches the emulation chip, see section C.6.2 on page C-14.

C.6.1.2 Using the XDS512RL On-Board Oscillator

To use the oscillator:

1) Locate jumper group JP25 and make sure there is a jumper across pins 1 and 2. This indicates that the clock source is taken from on board the XDS512RL.

2) Make sure the oscillator is installed on the XDS512RL at the position labeled U21. If the oscillator is has been removed previously, reinstall it.

3) Make sure the there is no cable attached to connector J2. J2 should only be used to provide an external clock source.

The XDS512RL on-board oscillator operates at 10MHz. To modify the frequency of the clock signal before it reaches the emulation chip, see section C.6.2 on page C-14.

C.6.1.3 Using an External SMA Clock Source

To use an external SMA clock source, follow these steps:

1) Locate jumper group JP25 and make sure there is a jumper across pins 1 and 2. This indicates that the clock source is taken from on board the XDS512RL.

2) Remove the oscillator if it is installed (at position U21 on the XDS512RL).

3) Use a cable to attached your external clock source to connector J2 on the XDS512RL.

To modify the frequency of the clock signal before it reaches the emulation chip, see section C.6.2 on page C-14.

C.6.2 Modifying the Clock Frequency

Once you have set up a clock source, you can modify the frequency of the clock signal before it reaches the emulation chip. You can do either or both of the following:

- ❏ Use the divide-by-2 mode to divide the clock frequency in half
- ❏ Use the PLL to multiply the clock frequency by preset factors

To produce the frequency you need, you might only need one of these features, or you might need to halve the source-clock frequency and then increase it with the PLL (phase-lock loop).

C.6.2.1 Dividing the Clock Frequency in Half

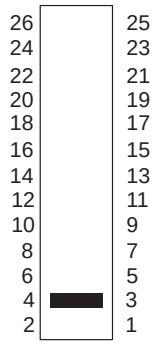
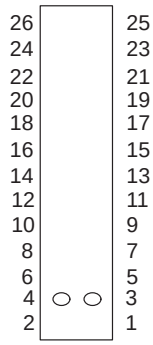
Once you have selected a clock source, you can use the divide-by-2 mode to divide the the frequency in half. You must turn the divide-by-2 mode on or off with JP25 pins 3 and 4 (JP25:3–4). Table C–12 identifies the pins, and Table C–13 shows how to use them. (For a complete pin description for JP25, see section C.6.4 on page C-19.)

After the frequency has been divided, it can be sent directly to the emulation chip, or it can be sent first to the phase-lock loop (PLL) to be multiplied.

Table C–12. JP25 Pins 3 and 4

Pin(s)	Signal	Description
3	DIV2SELn	Divides clock in half
4	GND	Ground

Table C–13. Turning the Divide-By-2 Mode On or Off With JP25:3–4

Divide-By-2 Mode On or Off?	JP25:3–4 Setting
On. The signal coming from the clock source is divided by 2 before being passed to the PLL or the emulation chip.	
Off. The signal is passed unchanged to the PLL or the emulation chip.	

C.6.2.2 Multiplying the Clock Frequency With the PLL

The PLL (phase-lock loop) allows you to apply a multiplication factor to the clock frequency before it is sent to the emulation chip. If the divide-by-2 mode is on, the source-clock frequency is already divided by 2 when it reaches the PLL.

To use the PLL, follow these steps:

1) Enable the PLL. Table C–14 (page C-16) shows how to enable or disable the PLL using JP26 pins 1, 2, 3, and 4. Table C–15 identifies these pins.

2) Select a PLL mode. JP25 pin sets 5–6, 7–8, and 9–10 determine the PLL mode as shown in Table C–16 (page C-17). If you are using the on-board oscillator, see Table C–17 for the frequency associated with the selected PLL mode. The pins are identified in Table C–18.

Table C–14. Enabling or Disabling the PLL With JP26:1–2 and JP26:3–4

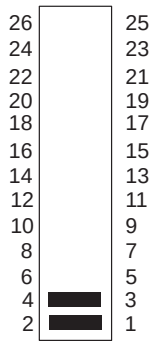
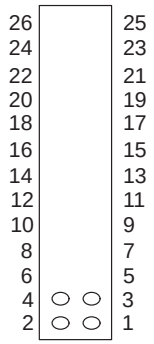
PLL Enabled or Disabled?	Default setting
Enabled. The PLL will multiply the clock signal by the chosen factor before passing it on the emulation chip.	
Disabled. The source clock signal is not affected by the PLL.	

Table C–15. JP26 Pins 1 Through 4

Pin(s)	Signal	Description
1	PLLSELn	To enable the the PLL, you must jumper pin sets 1–2 and 3–4.
2	GND	Ground
3	MODE	To enable the the PLL, you must jumper pin sets 1–2 and 3–4.
4	GND	Ground

Table C–16. PLL Multiplication Factors

PLL Mode Setting			Multiplication Factor	
JP25:9–10	**JP25:7–8**	**JP25:5–6**	**Divide-By-2 Off**	**Divide-By-2 On**
Jumper	Jumper	Jumper	x2 (default)	x1
Jumper	Jumper	Empty	x3	x1.5
Jumper	Empty	Jumper	x4	x2
Jumper	Empty	Empty	x5	x2.5
Empty	Jumper	Jumper	x6	x3
Empty	Jumper	Empty	x7	x3.5
Empty	Empty	Jumper	x8	x4
Empty	Empty	Empty	x10	x5

Table C–17. Frequencies Available When the 10-MHz Oscillator is the Clock Source

PLL Mode Setting			Input Clock Frequency	
JP25:9–10	**JP25:7–8**	**JP25:5–6**	**Divide-By-2 Off**	**Divide-By-2 On**
Jumper	Jumper	Jumper	20 MHz (default)	10 MHz
Jumper	Jumper	Empty	30 MHz	15 MHz
Jumper	Empty	Jumper	40 MHz	20 MHz
Jumper	Empty	Empty	50 MHz	25 MHz
Empty	Jumper	Jumper	60 MHz	30 MHz
Empty	Jumper	Empty	70 MHz	35 MHz
Empty	Empty	Jumper	80 MHz	40 MHz
Empty	Empty	Empty	100 MHz	50 MHz

Table C–18. JP25 Pins 5 Through 10

Pin(s)	Signal	Description
5	PLLMODE0	Least significant PLL mode bit
6	GND	Ground
7	PLLMODE1	Middle PLL mode bit
8	GND	Ground
9	PLLMODE2	Most significant PLL mode bit
10	GND	Ground

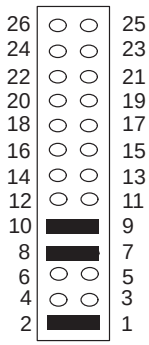
C.6.3 An Example of Setting Up a Clock

Suppose you want a 30-MHz clock signal for the emulation chip, and you want to use the on-board oscillator as the clock source. The oscillator provides a 10-MHz signal. To deliver 30 MHz to the emulation chip, you can use the PLL to multiply the oscillator's output by 3. The divide-by-2 mode must be turned off so that the frequency is not reduced to 15 MHz. The following procedure would prepare the XDS512RL for this clock scenario:

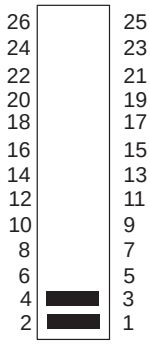
1) Make sure the oscillator is installed on the XDS512RL at the position labeled U21. If the oscillator is has been removed previously, reinstall it.

2) Make sure the there is no cable attached to connector J2. J2 should only be used to provide an external clock source.

3) Prepare jumper group JP25 as follows:

JP25	Description
	The jumper on pin set 1–2 selects an on-board clock source (either the XDS512RL oscillator or a signal connected to connector J2). The pin set 3–4 is not jumpered; the divide-by-2 mode is off. The jumper combination on pin sets 5–6, 7–8, and 9–10 program the PLL to multiply the clock-source signal by 3. Pins 11–26 are reserved for test purposes. Do not use these pins. (For descriptions of all the JP25 pins, see section C.6.4 on page C-19.)

4) Enable the PLL by jumpering JP26 pin sets 1–2 and 3–4.

JP26:1–2 and JP26:3–4	Description
	The PLL is enabled. (For descriptions of all the JP26 pins, see section C.6.4 on page C-19.)

C.6.4 Complete Pin Listings for JP25 and JP26

Table C–19 identifies all the pins of JP25. Pins 1–10 enable you to configure the clock for the emulation chip on board the XDS512RL. Do not use pins 11–26. The default jumper setting for JP25 is shown in Table C–26 on page C-27.

Table C–19. JP25 Pins

Pin(s)	Signal	Description
1	CLKSELn	Is used to select a target or non-target clock source.
2	GND	Ground
3	DIV2SELn	Divides clock in half
4	GND	Ground
5	PLLMODE0	Least significant PLL mode bit
6	GND	Ground
7	PLLMODE1	Middle PLL mode bit
8	GND	Ground
9	PLLMODE2	Most significant PLL mode bit
10	GND	Ground
11–18	Test switches	Never jumper these four pin sets. They are reserved for test purposes.
19–26	Test points	Never jumper these pins. They are reserved for test purposes.

Table C–20 identifies the pins of JP26. Use pin sets 1–2 and 3–4 to enable or disable the phase-lock loop (PLL). Jumper pin sets 5–6, 7–8, 9–10, and 11–12 to prevent noise they would otherwise add to the system. Use pin set 13–14 to change the location of the interrupt vectors. Pins 15 through 26 are reserved for test purposes. The default jumper setting for JP26 is shown in Table C–27 on page C-27.

Table C–20. JP26 Pins

Pin(s)	Signal	Description
1	PLLSELn	To enable the the PLL, you must jumper pin sets 1–2 and 3–4.
2	GND	Ground
3	MODE	To enable the the PLL, you must jumper pin sets 1–2 and 3–4.
4	GND	Ground
5	B0	You must jumper pins 5 and 6 for noise reduction.
6	GND	Ground
7	B1	You must jumper pins 7 and 8 for noise reduction.
8	GND	Ground
9	B2	You must jumper pins 9 and 10 for noise reduction.
10	GND	Ground
11	B3	You must jumper pins 11 and 12 for noise reduction.
12	GND	Ground
13	VMAP	Interrupt vector map signal. Pins 13 and 14 determine the program-address range used by the TMX320C2700–E1 when it fetches interrupt vectors (see section C.5 on page C-10).
14	GND	
15, 16	Test switch	Keep this pin set jumpered (default). This pin set is reserved for test purposes.
17–26	Test points	Never jumper these pins. They are reserved for test purposes.

CAUTION

Do not remove the jumper from JP26 pins 15 and 16. Removing the jumper will cause improper operation of the XDS512RL.

C.6.5 Removing the XDS512RL Oscillator

If you are planning to use an external clock source (attached at connector J2), you must remove the oscillator from the XDS512RL before you power up your emulation system.

C.6.6 Reinstalling the XDS512RL Oscillator

If the oscillator has been removed from the XDS512RL (at position U21) and you are planning to use the oscillator as a clock source for your emulation chip, you must reinstall the oscillator before you power up your emulation system.

C.7 Routing Target Signals to the TLA704

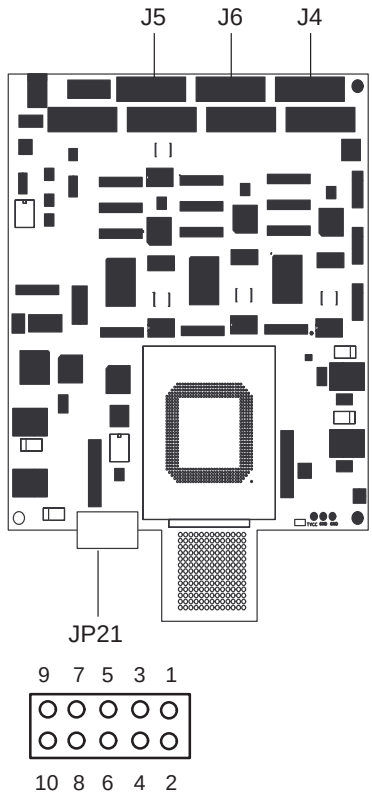
JP21 provides four signals (PROBE0–PROBE3) that let you route target system signals through MICTOR™ connectors to the TLA704. Table C–21 defines the pins of JP21. Figure C–3 shows the locations of JP21 and the MICTOR connectors (J4, J5, and J6).

Table C–21. JP21 Pins

Pin	Description
1 (PROBE0)	A signal connected to PROBE0 is transmitted to the TLA704 by way of MICTOR connector J6.
2 (PROBE1)	A signal connected to PROBE1 is transmitted to the TLA704 by way of MICTOR connector J6.
3 (PROBE2)	A signal connected to PROBE2 is transmitted to the TLA704 by way of MICTOR connector J5.
4 (PROBE3)	A signal connected to PROBE3 is transmitted to the TLA704 by way of MICTOR connector J4.
5	Ground
6	Ground
7	Reserved. Do not use this pin.
8	Reserved. Do not use this pin.
9	Reserved. Do not use this pin.
10	Reserved. Do not use this pin.

Note: MICTOR is a trademark of AMP Incorporated.

Figure C–3. JP21 and MICTOR Connectors J4, J5, and J6



C.8 Accessing Range-Logic Signals at XDS512RL Pins

Figure C–4 shows the location and pins of jumper groups JP4, JP5, and JP6. The pins for these jumper groups are defined in Table C–22. All these pins except the ground pins are for outputs from the range logic that is on board the XDS512RL.

For each range-logic bank (program, read, and write), there are eight 32K address blocks, labeled 0–7. As you create your trace file, the BTT application assigns events to these range-logic blocks. When an event assigned to block 4, 5, 6, or 7 occurs, a signal is sent to the TLA704 and an inverted version of that signal is sent to the corresponding XDS512RL pin. For example, suppose an event has been assigned to block 5 in the read bank. If the event occurs, a ReadHigh signal is sent to the TLA704, and an inverted version of the ReadHigh signal is sent to the Read5 pin on JP5.

Figure C–4. JP4, JP5, and JP6

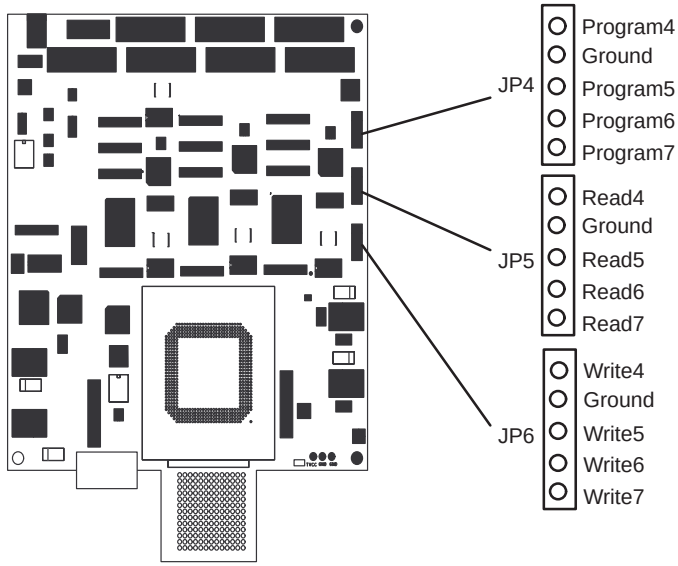


Table C–22. The Pins of JP4, JP5, and JP6

Jumper Group	Pin	Signal	Description
JP4	1	Program4	When low, Program4 indicates the occurrence of an event tied to range-logic block 4 in the program bank.
	2	Ground	Ground
	3	Program5	When low, Program5 indicates the occurrence of an event tied to range-logic block 5 in the program bank.
	4	Program6	When low, Program6 indicates the occurrence of an event tied to range-logic block 6 in the program bank.
	5	Program7	When low, Program7 indicates the occurrence of an event tied to range-logic block 7 in the program bank.
JP5	1	Read4	When low, Read4 indicates the occurrence of an event tied to range-logic block 4 in the read bank.
	2	Ground	Ground
	3	Read5	When low, Read5 indicates the occurrence of an event tied to range-logic block 5 in the read bank.
	4	Read6	When low, Read6 indicates the occurrence of an event tied to range-logic block 6 in the read bank.
	5	Read7	When low, Read7 indicates the occurrence of an event tied to range-logic block 7 in the read bank.
JP6	1	Write4	When low, Write4 indicates the occurrence of an event tied to range-logic block 4 in the write bank.
	2	Ground	Ground
	3	Write5	When low, Write5 indicates the occurrence of an event tied to range-logic block 5 in the write bank.
	4	Write6	When low, Write6 indicates the occurrence of an event tied to range-logic block 6 in the write bank.
	5	Write7	When low, Write7 indicates the occurrence of an event tied to range-logic block 7 in the write bank.

C.9 Summary of Default Jumper Settings

When you receive your XDS512RL, the some jumper groups have (removeable) jumpers that were put on at the factory. The way each jumper group is set at the factory is that group's *default jumper setting*. The following are the default settings for jumper groups JP2, JP20, JP23, JP25, and JP26, in that order.

Table C–23. Default Jumper Setting for JP2

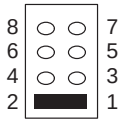
JP2 Default Setting	Operating Mode Selected
	*Stand-alone hold-in-reset.* The emulation chip is held in reset after power-up. You can use this mode to hold the emulation chip in reset mode while you download your code. This mode is for operation without a target.

Table C–24. Default Jumper Setting for JP20

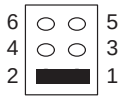
JP20 Default Setting	Reset Mode Selected
	Target OFF

Table C–25. Default Jumper Setting for JP23

JP23 Default Setting	Power Mode Selected
	*Power shared.* The target system and the XDS510RL share the XDS512RL power source (and they share the same ground).

Table C–26. Default Jumper Setting for JP25

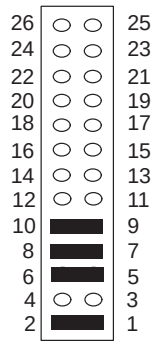
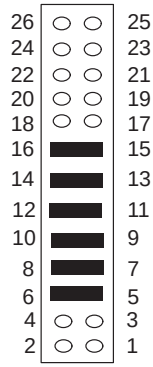
JP25 Default Setting	Description
	The jumper on pin set 1–2 selects an on-board clock source (either the XDS512RL oscillator or a signal connected to connector J2). Pin set 3–4 is not jumpered; the clock-source signal is not divided by 2. The jumpers on pin sets 5–6, 7–8, and 9–10 program the PLL to multiply the clock-source signal by 2.

Table C–27. Default Jumper Setting for JP26

JP26 Default Setting	Description
	The pins sets 1–2 and 3–4 are not jumpered; the PLL is disabled. The pin sets 5–6, 7–8, 9–10, and 11–12 are jumpered for noise reduction. The jumper on pin set 13–14 maps the TMX320C2700–E1 interrupt vectors to program-space addresses 0x00 0000–0x00 003F. Pin set 15–16 must be kept jumpered at all times.

Appendix D

Glossary

B

BTT: *Breakpoint, tracing, and timing*. A software tool that adds breakpoint, tracing, and timing functionality to a source code debugger.

C

C27xsetup.tla file: A default instrument setup file for the C27x support package.

E

EISA: *Extended Industry Standard Architecture*. A standard for PC buses

G

GUI: *Graphical User Interface*.

I

ISA: *Industry Standard Architecture*. A subset of the EISA standard.

J

JTAG cable: The cable that attaches the XDS510 to your target system.

E

emulation device: A device that connects to the XDS512RL. An emulation device is a specialized device that performs the same function as your target device but includes features for gathering debugging information.

T

target system: The system you want to debug. This can be your actual system, a test system you have created, or an XDS512 with an emulation device attached to it.

X

XDS510: An emulator controller board for use in an IBM-compatible PC. The XDS510 enables your debugger to communicate with the target system.

XDS512RL emulator kit: The XDS512 emulator kit consists of the XDS510 emulator controller board, a JTAG cable, P6434 high-density probes, Tektronix logic analyzer TLA704, 'C27x support package, and a 'C27x emulation device.

XDS512RL emulator: A board used to emulate a target system. The XDS512 is connected to an XDS510 or XDS510PP and Tektronix logic analyzer. In turn, you can connect the XDS512 to a target system.

Index

A

B

C

D

E

F

G

I

J

M

P

Q

R

S

T

U

W

X