

User's Guide

Publication number E2493-97007
August 2001

For Safety information, Warranties, and Regulatory information, see the pages behind the index.

© Copyright Agilent Technologies 1994-2001
All Rights Reserved

Logic Analysis Support for the ARM7/ARM9

Solutions for the ARM7/ARM9—At a Glance

This book documents the ARM7/ARM9 Inverse Assembler

The Agilent Technologies E9595A inverse assembler, in conjunction with an Agilent Technologies logic analyzer, allows you to view ARM® assembly instructions that are executing in your target system.

The inverse assembler is compatible with a variety of 8-bit, 16-bit or 32-bit ARM core, AMBA (Advanced Microcontroller Bus Architecture) ASB (Advanced System Bus) and AHB (Advanced High-performance Bus) systems. The inverse assembler can be configured to work with the signals that are available for probing.

The inverse assembler model number is Agilent Technologies E9595A Option 001 when ordered alone. It is identified as “Agilent Technologies E2493A” in the Setup Assistant.

If You Purchased an Emulation Solution

The E9495A/B emulation solution lets you use an HP or Agilent Technologies logic analysis system to debug and characterize a variety of ARM target systems. The emulation solution is a bundled product consisting of an inverse assembler, an emulation module (and its cables and adapters), and the Agilent Technologies B4620B source correlation tool set. This solution is designed to be used with an Agilent Technologies 16700-series logic analysis system.

For more information on an emulation solution

The *E5900A/B Option 300 Emulation for the ARM7/ARM9 User's Guide* describes setting up and using the emulation probe and emulation module.

Information about using the logic analysis system with the emulation probe/module can be found in Chapter 10, “Coordinating Logic Analysis with Processor Execution,” beginning on page 231 of *this* manual.

Additional Equipment Included in an Emulation Solution

Emulation Module and Emulation Probe

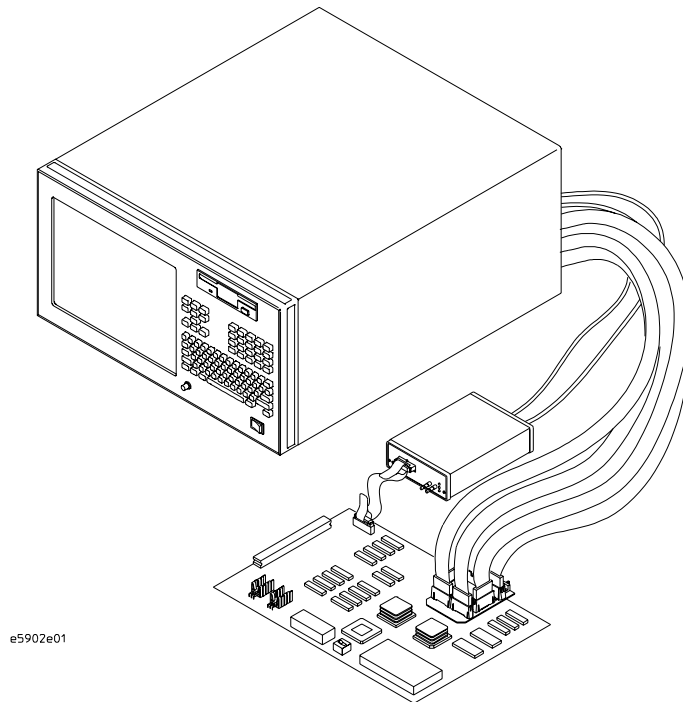
The emulation module plugs into your Agilent Technologies 16700-series logic analysis system frame, and the emulation probe connects to the emulation module and a debug port on your target system. The emulation probe lets you use the target processor's built-in background debugging features, including run control and accessing registers and memory. A high-level source debugger can use the emulation probe/module to debug code running on the target system.

Information about using the logic analysis system with the emulation probe can be found in Chapter 10, "Coordinating Logic Analysis with Processor Execution," beginning on page 231 of *this* manual. The *Emulation for the ARM7/ARM9 User's Guide* describes setting up and using the emulation probe and emulation module.

Source Correlation Tool Set

The Agilent Technologies B4620B source correlation tool set lets you set up logic analyzer triggers based on source code, and it lets you view the source code associated with signals captured by the logic analyzer.

Emulation Solution



In This Book

This book documents the following products:

Processor supported	Product ordered	Includes
ARM7/ARM9	Agilent Technologies E9595A Option #001 inverse assembler only	Inverse assembler

Related equipment

The following equipment is included in the ARM7/ARM9 emulation solution.

Processor supported	Product ordered	Includes
ARM7/ARM9	Agilent Technologies E9495A/B Option #001 emulation solution	Inverse assembler, emulation probe, emulation module, and B4620B Source Correlation Tool Set

Tips To Save You Time

Use the Setup Assistant

Click here to use the Setup Assistant, the menu-driven guide for connecting your system. It will automatically load the correct configuration files. See page 22.

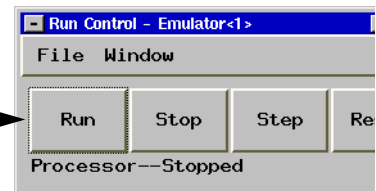


Use the appropriate Run button



Click here to start a measurement.

If your system includes an emulation probe/module, click here to run the target microprocessor.



Additional Information Sources

Newer editions of this manual may be available. Contact your local Agilent Technologies representative.

If you have a probing adapter, the instructions for connecting the probe to your target system are in the **Probing Adapter** documentation.

Application notes may be available from your local Agilent Technologies representative or on the World Wide Web at:

<http://www.agilent.com/find/logicanalyzer>

If you have an Agilent 16700 series logic analysis system with an emulation probe/module, the **online help** for the Emulation Control Interface has additional information on using the emulation module. Also, see the emulation manual included with your emulation probe/module.

The **measurement examples** include valuable tips for making emulation and analysis measurements. You can find the measurement examples under the system help in your Agilent 16700-series logic analysis system.

Contents

Additional Equipment Included in an Emulation Solution	3
Use the Setup Assistant	5
Use the appropriate Run button	5

1 Equipment and Requirements 19

Setup Checklist	21
Setup Assistant	22
Setup Flow Diagram	23
Inverse Assembler	24
Equipment supplied	24
Additional Equipment Required	25
Additional equipment supported	25
Logic Analyzer Requirements	26
To determine whether your logic analyzer is compatible	26
Logic Analyzer Descriptions	27
ARM bus system type and Agilent logic analyzer model compatibility	28
Number of logic analyzer pods available (per analyzer)	29
Number of logic analyzer pods required	30
Logic analyzer software version requirements	31
Emulation Solution	32
Emulation solution	32

2 Preparing the Target System for ARM Core and AMBA ASB 33

Choosing a Connector Type	35
High-Density Connectors	37
Medium-Density Connectors	40
Signal Requirements for ARM Core or AMBA ASB Inverse Assembly	42
Required Signals	43
Strongly Recommended Signals	45
Recommended Signals	46
Optional Signals	48
ARM7/ARM9 Core Signal Details	49
ARM7/ARM9 Core to AMBA ASB Signal Conversion	50
ARM7/ARM9 Inverse Assembler STAT Pod Signals	51
Signal-to-Connector Mappings - ARM Core	54
ARM core signal connector mappings for 8-bit bus systems	55
ARM core signal connector mappings for 8-bit bus systems (continued)	56
ARM core signal connector mappings for 8-bit bus systems (continued)	57
ARM core signal connector mappings for 8-bit bus systems (continued)	58
ARM core signal connector mappings for 16-bit bus systems (reduced address mode)	59
ARM core signal connector mappings for 16-bit bus systems (reduced address mode) (continued)	60
ARM core signal connector mappings for 16-bit bus systems (reduced address mode) (continued)	61
ARM core signal connector mappings for 16-bit bus systems (reduced address mode) (continued)	62
ARM core signal mappings for 16/32-bit bus systems	63
ARM core signal connector mappings for 16/32-bit bus systems (continued)	64

Contents

ARM core signal connector mappings for 16/32-bit bus systems (continued)	65
ARM core signal connector mappings for 16/32-bit bus systems (continued)	66
ARM core signal connector mappings for 16/32-bit bus systems (continued)	67
ARM core signal connector mappings for 16/32-bit bus systems (continued)	68
Signal-to-Connector Mappings - AMBA ASB 69	
AMBA ASB signal connector mappings for 8-bit bus systems	70
AMBA ASB signal connector mappings for 8-bit bus systems (continued)	71
AMBA ASB signal connector mappings for 8-bit bus systems (continued)	72
AMBA ASB signal connector mappings for 8-bit bus systems (continued)	73
AMBA ASB signal connector mappings for 16-bit bus systems (reduced address mode)	74
AMBA ASB signal connector mappings for 16-bit bus systems (reduced address mode) (continued)	75
AMBA ASB signal connector mappings for 16-bit bus systems (reduced address mode) (continued)	76
AMBA ASB signal connector mappings for 16-bit bus systems (reduced address mode) (continued)	77
AMBA ASB signal connector mappings for 16/32-bit bus systems	78
AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)	79
AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)	80
AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)	81
AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)	82
AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)	83
Designing a JTAG Connector into Your Target System	84

3 Preparing the Target System for AMBA AHB 85

High-Density Connectors	87
Signal Requirements for ARM AMBA AHB Inverse Assembly	90
Single Master and Multiple Master Configurations	90
Single Master Configuration	91
Required Signals	91
Optional Signals	92
Other Signals Not Used for Inverse Assembly	92
ARM AMBA AHB Signal Details — Single Master	93
Multiple Master Configuration	95
Required Signals	95
Optional AHB Signals	96
Other AHB Signals Not Used for Inverse Assembly	96
ARM AMBA AHB Signal Details — Multiple Masters	97
ARM AMBA AHB Inverse Assembler STAT Pod Signals	100
Signal-to-Connector Mappings - AMBA AHB	102
Signal-to-Connector Mappings — Single Master Configuration	102
Signal-to-Connector Mappings — Multiple Master Configuration	107
Designing a JTAG Connector into Your Target System	112

4 Setting Up the Logic Analysis System 113

Power-on/Power-off Sequence	114
To power on Agilent 16700-series logic analysis systems	114
To power on all other logic analyzers	115
To power off	115

Installing Logic Analyzer Modules	116
Installing the Emulation Module	117
Installing Software	118
Installing and loading	118
What needs to be installed	118
To install the software from CD-ROM	119

5 Probing the Target System 121

Connecting the Logic Analyzer to the Target System	122
Connecting the Logic Analyzer to the Target System	123
To connect to a 16715/16/17/18/19A or 16750/51/52A logic analyzer (two cards)	124
To connect to a 16715/16/17/18/19A or 16750/51/52A logic analyzer (one card)	125
To connect to a 16710/11/12A logic analyzer (two card)	126
To connect to a 16710/11/12A logic analyzer (one card)	127
To connect to a 16603A logic analyzer	128
To connect to a 16602A logic analyzer	129
To connect to a 16601A logic analyzer	130
To connect to a 16600A logic analyzer	131
To connect to a 16554/55/56/57 logic analyzer (two-card)	132
To connect to a 16554/55/56/57 analyzer (one-card)	133
To connect to a 16550A logic analyzer (two card)	134
To connect to a 16550A logic analyzer (one card)	135
To connect to a 1671A/D/E logic analyzer	136
To connect to a 1670A/D/E logic analyzer	137
To connect to a 1661A/AS/C/CS/E/ES/EP logic analyzer	138
To connect to a 1660A/AS/C/CS/E/ES/EP logic analyzer	139

6 Configuring the 16700-series Logic Analysis System 141

Configuring 16700-series Logic Analysis Systems	142
To load configuration files (and the inverse assembler) from hard disk—16700-series logic analysis systems	143
To load configuration files (and the inverse assembler) from floppy disk—16700-series logic analysis systems	144
To list software packages that are installed (16700-series logic analysis system)	145
ARM7 core analysis configuration files	146
ARM 7 AMBA ASB analysis configuration files	148
ARM9 AMBA ASB analysis configuration files	150
ARM AMBA AHB analysis configuration files	152
Inverse Assembler Modes of Operation	153
State mode	153
To change to timing mode	153
Disabling the cache	153
To use the Invasm menu	154
Loading the Inverse Assembler	154
Unloading the Inverse Assembler	154
Setting the Inverse Assembler Preferences	155
To set the memory map preferences — ARM core and AMBA ASB	156
Signals Dialog — ARM core and AMBA ASB	158
To set AMBA AHB memory map preferences and signal information	161
Symbols	163
Predefined ARM Symbols	164
Object File Symbols	165
Requirements	165
To use object file symbols in the 16700	166

7 Configuring the 1660A/1670A/16500B/C-Series Logic Analyzer 169

Analyzing the ARM7/ARM9 with a 1660/1670/16500B/C Logic Analyzer 170

Configuring the Inverse Assembler 170

Making Data Measurements 170

Analyzer Modes 171

Configuring Logic Analyzer IA Menus 172

Configuring the Logic Analysis System 181

To load configuration and inverse assembler files—Agilent 16700 logic analysis systems 182

To load configuration files—other logic analyzers 183

8 Capturing Processor Execution 185

Trigger sequence 187

Predefined trigger terms 187

To use predefined trigger terms 188

To view the definition of the trigger term 190

To Set Up Logic Analyzer Triggers 191

Triggering on Symbols and Source Code 193

To avoid triggering on prefetched instructions 193

To correlate relocatable code using the address offset 194

Triggering ARM Data on the 1660/70-series logic analyzers 195

Contents

Making Common Measurements Using the
Agilent 16700 Logic Analysis System 197

Example 1: Setting a trigger for a specific address 198

Example 2: Triggering on a write to a variable 199

Example 3: Triggering on a 16-bit write to variable 200

Example 4: Setting a trigger at exit of debug mode 203

Example 5: Store qualifying wait states 204

Making Common Measurements Using All Other HP/Agilent Logic
Analyzers 205

Example 1: Setting a trigger for a specific address 206

Example 2: Triggering on a write to a variable 207

Example 3: Triggering on a 16-bit write to variable 208

Example 4: Setting a trigger at exit of debug mode 212

Example 5: Store qualifying wait states 213

9 Displaying Captured Processor Execution 215

Viewing ARM Trace Data 216

Display Filtering 218

Display Filtering Dialog—ARM Core and AMBA ASB 219

Display Filtering Dialog—AMBA AHB 222

Displaying Source Code 225

Inverse Assembler Generated PC (Software Address) Label 227

Access to Source Code Files 228

Viewing ARM core or AMBA ASB Trace Data on the 1660/70-series logic
analyzers 229

10 Coordinating Logic Analysis with Processor Execution 231

- What are some of the tools I can use? 232
- Which assembly-level listing should I use? 233
- Which source-level listing should I use? 233
- Where can I find practical examples of measurements? 233

- Triggering the Emulation Module from the Analyzer 234
 - To stop the processor when the logic analyzer triggers on a line of source code (Source Viewer window) 234
 - To stop the processor when the logic analyzer triggers (Intermodule window) 236
 - To minimize the "skid" effect 237
 - To configure the availability of DBGACK and DBGRQ 238
 - To stop the analyzer and view a measurement 239

- Tracing until the processor halts 240
 - To capture a trace before the processor halts 241

- Triggering the Logic Analyzer from the Emulation Module 242
 - The emulation module trigger signal 242
 - Group Run 243
 - Debuggers can cause triggers 245
 - To trigger the analyzer when the processor halts - timing mode 246
 - To trigger the analyzer when the processor reaches a breakpoint 248

11 General-Purpose ASCII (GPA) Symbol File Format 251

General-Purpose ASCII (GPA) Symbol File Format	252
GPA Record Format Summary	254
SECTIONS	256
FUNCTIONS	257
VARIABLES	258
SOURCE LINES	259
START ADDRESS	260
Comments	260

12 Troubleshooting 261

Logic Analyzer Problems	263
Intermittent data errors	263
Unwanted triggers	264
No activity on activity indicators	264
No trace list display	264
Capacitive loading	265
Inverse Assembler Problems	266
No inverse assembly or incorrect inverse assembly	266
Inverse assembler will not load or run	267
Inverse Assembler Error Messages	268
“IA Error - Address not in memory map”	268
“IA Error - Search limited by depth”	268
“Inverse Assembler Not Found”	268
“No Configuration File Loaded”	269
“Selected File is Incompatible”	269
“Slow or Missing Clock”	269

Contents

“Waiting for Trigger”	270
Understanding the Impact of ARM Signal Availability	271
LDR PC and LDM PC instructions without nMREQ	271
Problems with literal pools and no nOPC signal	272
SWP instructions without nRW, nWAIT and nMREQ	273
Missing MAS[1] and incorrect memory map entries	273
DMA accesses showing up as OPCODES or DATA	274
Store qualification not storing states needed by the inverse assembler for proper operation	274
Other problem scenarios	275
The inverse assembler trace looks like there are several instructions being decoded for the same address	275
Some data fetches in the trace listing seem to be shown as instructions	276
A data state is immediately repeated by another data state or an instruction fetch	277
State symbols (*) for the unused prefetch states don’t seem be correct all of the time	277
An extra extension state for an instruction fetch is seen when using reverse memory controller	278
Intermodule Measurement Problems	279
An event wasn’t captured by one of the modules	279
“No Configuration File Loaded”	280
“Selected File is Incompatible”	280
“Slow or Missing Clock”	280
“Waiting for Trigger”	281

13 Specifications and Characteristics 283

Glossary	285
-----------------	------------

Contents

Equipment and Requirements

Chapter 1: Equipment and Requirements

This chapter describes:

- Setup Checklist
- Setup Assistant
- Setup Flow Diagram
- Equipment supplied with the inverse assembler
- Additional equipment required
- Logic analyzer requirements
- Logic analyzer descriptions
- ARM system bus type and logic analyzer compatibility
- Number of logic analyzer pods available
- Number of logic analyzer pods required
- Logic analyzer software version requirements
- Emulation solution

Setup Checklist

Follow these steps to connect your equipment:

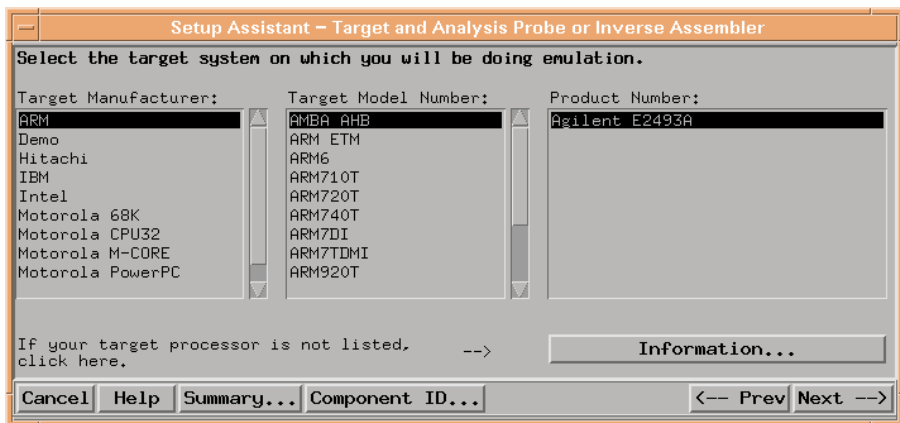
- Check that you received all of the necessary software. See page 24.
- If you need to install an emulation module in an Agilent 16700-series logic analysis system, see your emulation manual.
- Install the software. See page 118.
- If you have an Agilent 16700-series logic analysis system, use the Setup Assistant to help you connect and configure your system. See page 22.

Setup Assistant

The Setup Assistant is an online tool for connecting and configuring your logic analysis system for microcontroller and bus analysis. The Setup Assistant is available on the Agilent 16700-series logic analysis systems. You can use the Setup Assistant in place of the connection and configuration procedures provided in this manual.

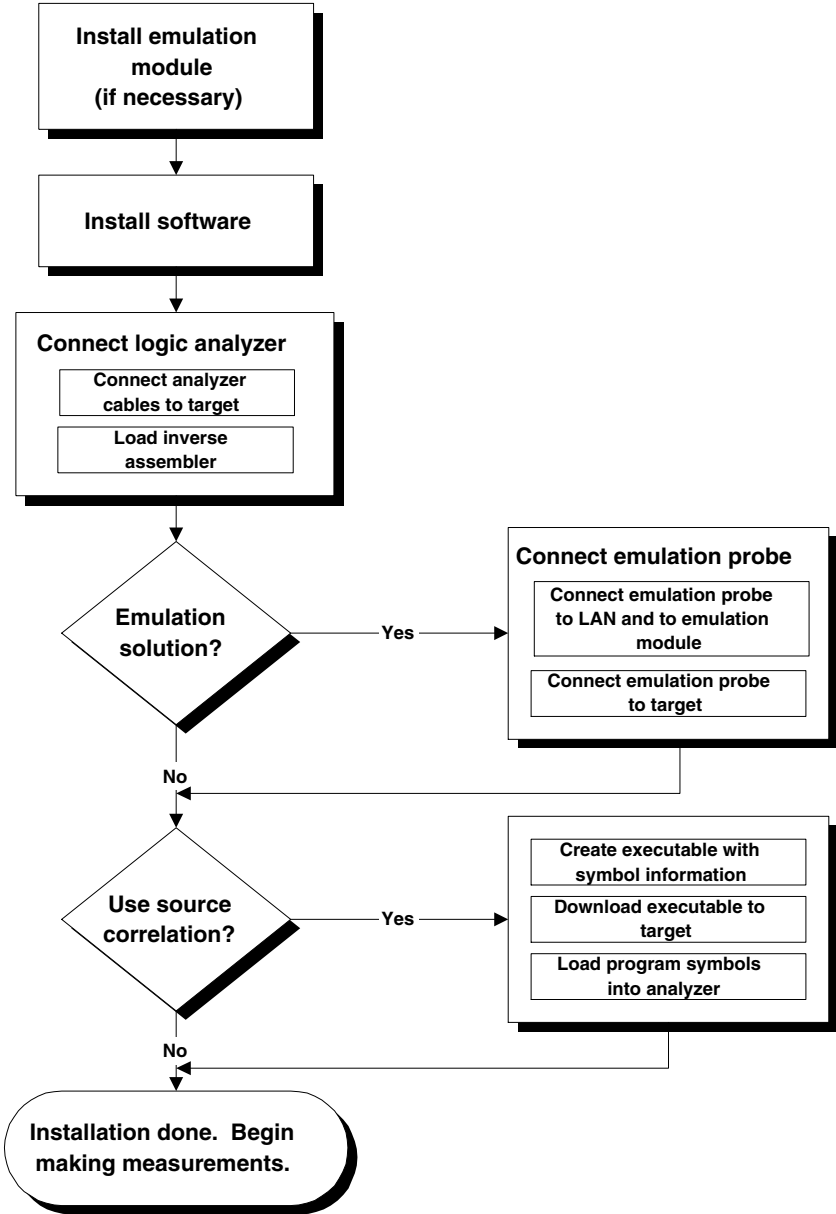
This menu-driven tool will guide you through the connection procedures for connecting the logic analyzer to an analysis probe, an emulation module, or other supported equipment. It will also guide you through connecting the logic analyzer pods to connectors on the target system.

Start the Setup Assistant by selecting  in the system window.



If you ordered this product with your Agilent 16700-series logic analysis system, the logic analysis system has the latest software installed, including support for this product.

Setup Flow Diagram



E2498F02.VSD

Inverse Assembler

This section lists equipment supplied with the inverse assembler and equipment requirements for using the inverse assembler.

Equipment supplied

The inverse assembler consists of:

- The ARM/AMBA ASB and AHB inverse assembler software (with configuration files) on CD-ROM. Includes versions for normal and reverse memory controllers.
- The ARM/AMBA ASB and AHB inverse assembler software (with configuration files) on 3.5-inch logic analyzer disks. Includes versions for normal and reverse memory controllers.
- This User's Guide.

Additional Equipment Required

In addition to the items listed above, the following is required to analyze an ARM target circuit:

- Connector headers on your target system which supply signals to the logic analyzer.
- Agilent Technologies termination adapter cables to attach your target system to a logic analyzer.
- One of the logic analyzers listed in the table on the following page.

Additional equipment supported

Agilent Technologies B4620B Source Correlation Tool Set.

The inverse assembler may be used with the Agilent Technologies B4620B Source Correlation Tool Set.

Logic Analyzer Requirements

Due to variations in design and implementation of ARM target systems and the wide variety of Agilent Technologies logic analyzers available, there are several inverse assembler configuration files. (Configuration files specify how the logic analyzer channels connect to the target system.)

Each target system bus configuration (e.g. ARM core, ARM AMBA AHB, etc.) requires a certain minimum number of logic analyzer pods (as listed on page 30) for meaningful inverse assembly. Also, some of the older logic analyzers can not be used for inverse assembly on the AMBA AHB bus.

To determine whether your logic analyzer is compatible

- 1** Find out the model number of your logic analyzer card (which is installed in the mainframe) by looking at the main system window of the logic analyzer. If you have an older logic analyzer and the model number is not displayed in the System window, you can refer to the table on page 27.
- 2** Determine whether the logic analyzer model number is compatible using the table on page 28.
- 3** Check that there are enough pods available:
 - Look up available pod count on page 29
 - Look up number of pods required on page 30

Logic Analyzer Descriptions

If you have a 16700-series logic analysis system, it shows logic analyzer card model numbers in the system window; you can ignore this table. Certain older logic analysis systems only show speed and memory depth information in the system window. The table below can be used to determine which model logic analyzer card is installed in older systems. Actual performance depends on configuration.

Logic Analyzer	Channel Count	State Speed	Timing Speed	Memory Depth
16752A	68/card	400 MHz	2 GHz	32 M states
16751A	68/card	400 MHz	2 GHz	16 M states
16750A	68/card	400 MHz	2 GHz	4 M states
16719A	68/card	333 MHz	2 GHz	32 M states
16718A	68/card	333 MHz	2 GHz	8 M states
16717A	68/card	333 MHz	2 GHz	2 M states
16716A	68/card	167 MHz	2 GHz	512 k states
16715A	68/card	167 MHz	2 GHz	2 M states
16712A	102/card	100 MHz	500 MHz	128 k states
16711A	102/card	100 MHz	500 MHz	32 k states
16710A	102/card	100 MHz	500 MHz	8 k states
16603A	68	100 MHz	125 MHz	64 k states
16602A	102	100 MHz	125 MHz	64 k states
16601A	136	100 MHz	125 MHz	64 k states
16600A	204	100 MHz	125 MHz	64 k states
16557D	68/card	135 MHz†	250 MHz†	2 M states
16556A	68/card	100 MHz	200 MHz	1 M states
16555D/56D	68/card	100 MHz	500/400 MHz	2 M states
16555A	68/card	110 MHz	250 MHz	1 M states
16554A	68/card	70 MHz	125 MHz	512 k states
16550A	102/card	100MHz	250 MHz	4 k states
1671A	102	70 MHz	125 MHz	64 k or 0.5 M
1670D/E 1671D/E	136/102	100 MHz	250 MHz	1 M states
1670A	136	70 MHz	125 MHz	64 k or 0.5 M
1661A/AS/C/CS/CP/E/ES/EP	102	100 MHz	250 MHz	4 k states
1660A/AS/C/CS/CP/E/ES/EP	136	100 MHz	250 MHz	4 k states

† The 16557D state and timing speeds decrease for four- or five-card configurations.

ARM bus system type and Agilent logic analyzer model compatibility

The following table shows whether a particular logic analyzer model is compatible with the inverse assembler when it is used on an ARM core, AMBA ASB, or AMBA AHB system.

Logic Analyzer	ARM core and AMBA ASB	AMBA AHB
16752A	Yes	Yes
16751A	Yes	Yes
16750A	Yes	Yes
16719A	Yes	Yes
16718A	Yes	Yes
16717A	Yes	Yes
16716A	Yes	Yes
16715A	Yes	Yes
16712A	Yes	Yes
16711A	Yes	Yes
16710A	Yes	Yes
16603A	Yes	No
16602A	Yes	No
16601A	Yes	No
16600A	Yes	No
16557D	Yes	Yes
16556A	Yes	Yes
16555D/56D	Yes	Yes
16555A	Yes	Yes
16554A	Yes	Yes
16550A	Yes	Yes
1671A	Yes	No
1670D/E 1671D/E	Yes	No
1670A	Yes	No
1661A/AS/C/CS/CP/E/ES/EP	Yes	No
1660A/AS/C/CS/CP/E/ES/EP	Yes	No

Number of logic analyzer pods available (per analyzer)

A logic analyzer mainframe system can generally house between one and five logic analyzer cards. Each card has a number of pods. Other logic analyzers, which are not logic analysis frame systems, have a fixed channel count and hence a fixed number of pods.

The following table shows the number of pods available based on logic analyzer model number and the number of cards installed (if the logic analyzer is installed in a logic analysis system mainframe).

Logic Analyzer Model Number	Number of logic analyzer Channels	Number of cards installed (if applicable)	Number of Pods (Based on number of cards)
16750/51/52A	68/card	2 (or more)	8 (or more)
16750/51/52A	68/card	1 card	4
16715/16/17/18/19A	68/card	2 (or more)	8 (or more)
16715/16/17/18/19/A	68/card	1 card	4
16710/11/12A	102	2 (or more)	12 (or more)
16710/11/12A	102	1 card	6
16603A	68	n/a	4
16602A	102	n/a	6
16601A	136	n/a	8
16600A	204	n/a	12
16554/55/56/57	68/card	2 (or more)	8 (or more)
16554/55/56/57	68/card	1 card	4
16550A	102/card	2 (or more)	12 (or more)
16550A	102/card	1 card	6
1671A/D/E	102	n/a	6
1670A/D/E	136	n/a	8
1661A/AS/C/CS/CP/E/ES/EP	102	n/a	6
1660A/AS/C/CS/CP/E/ES/EP	136	n/a	8

Number of logic analyzer pods required

The following table shows the number of logic analyzer pods required for inverse assembly on various types of ARM target systems.

There are two pods per connector on a logic analyzer card.

Target System Configuration	Minimum number of pods required
ARM core or AMBA ASB 8-bit data bus	4
ARM core or AMBA ASB 16/32 bit data bus	6
ARM core or AMBA ASB 16/32 bit data bus Reduced address mode	4
AMBA AHB single master Separate HRDATA and HWDATA buses	8
AMBA AHB multiple master Combine HRDATA and HWDATA to HRDATA bus	8

The HRDATA and HWDATA buses can be combined in which case both read and write transactions can be seen on the HRDATA bus. See “Setting the Inverse Assembler Preferences” on page 155.

Logic analyzer software version requirements

The logic analyzers must have software with a version number greater than or equal to those listed below to make a measurement with the inverse assembler. You can obtain the latest software at the following web site:

<http://www.agilent.com/find/logicanalyzer>

If your software version is older than those listed, load new system software with the higher version numbers before loading the inverse assembler software.

Logic Analyzer Software Version Requirements

Agilent Technologies Logic Analyzer	Minimum Logic Analyzer Software Version
16600A-series	The latest Agilent Technologies 16600A logic analyzer software version is on the CD-ROM shipped with this product.
1660A/AS Series	3.01
1660C/CS/CP/E/ES/EP Series	A.02.01
1670A/D/E Series	A.02.02
Mainframes*	
16700-series	The latest Agilent Technologies 16700 logic analyzer software version is on the CD-ROM shipped with this product.
16500C Mainframe	1.07
16500B Mainframe	3.14

* The mainframes are used with the Agilent Technologies logic analyzer cards.

Emulation Solution

If you ordered an emulation solution, you received an emulation probe, an emulation module and accessories (cables and power supply) which are described in the *E5900(A or B) Option 300 Emulation for the ARM7/ARM9 User's Guide*.

Emulation solution

The combination of an inverse assembler, an emulation module, and an Agilent 16700-series logic analysis system lets you both view ARM assembly instructions that are executing on your target system and use the target processor's built in JTAG debugging features.

You can use a debugger or the logic analysis system's Emulation Control Interface to configure and control the target processor and to download program code. You can use the Agilent Technologies B4620B Source Correlation Tool Set to analyze high-level source code using the logic analysis system.

Preparing the Target System for ARM
Core and AMBA ASB

This chapter describes the necessary connections between the target system and the logic analysis system for inverse assembly on an ARM core or AMBA ASB target system.

Because ARM circuits will vary with each design, it is important that you design headers into your target system for connection to a logic analyzer.

If your system already contains connector headers with incompatible pinouts, you can still connect to a logic analyzer using an adapter cable and General Purpose probes.

- When designing a system, choose a connector type shown in “Choosing a Connector Type” on page 35. Once you’ve decided on a connector type, you can design your ARM circuit board to contain the connector headers.
- When your target system contains medium-density or high-density connector headers, use Agilent Technologies cables or termination adapters to connect your target system to a logic analyzer.

Choosing a Connector Type

ARM core or AMBA ASB signals on your target system connect to a logic analyzer via connector headers that you build into the target circuit board.

NOTE:

You must choose either medium-density or high-density connectors and design your target system circuit board to contain the connector headers!

Items to consider when selecting a type of connector are:

- Board space required by the connector header
- Proper termination of the logic analyzer
- Proper connection of target signals to the logic analyzer pods

The medium-density and high-density connectors are summarized below.

	Medium-Density Connector	High-Density Connector
Pin Configuration	2 rows x 10 pins	2 rows x 19 pins
Header Part Number	3M 2520-6002	AMP 2-767004-2
Required Termination Adapter	Agilent Part 01650-63203	Agilent E5346A (one for every two logic analyzer pods)
Notes	Not recommended above 50 MHz	Each connector supports two logic analyzer pods

-
- The Agilent Technologies 01650-63203 termination adapter connects a logic analyzer pod to a medium-density connector header while providing the proper termination.
 - The Agilent Technologies E5346A is a Y-cable which connects two logic analyzer pods to one high-density connector header while providing the proper termination.

NOTE:

If a PC board already has medium-density or high-density connector headers attached, but the signal pinouts do not match the requirements as shown in the signal-to-connector mappings in this chapter, you can still connect to a

Chapter 2: Preparing the Target System for ARM Core and AMBA ASB

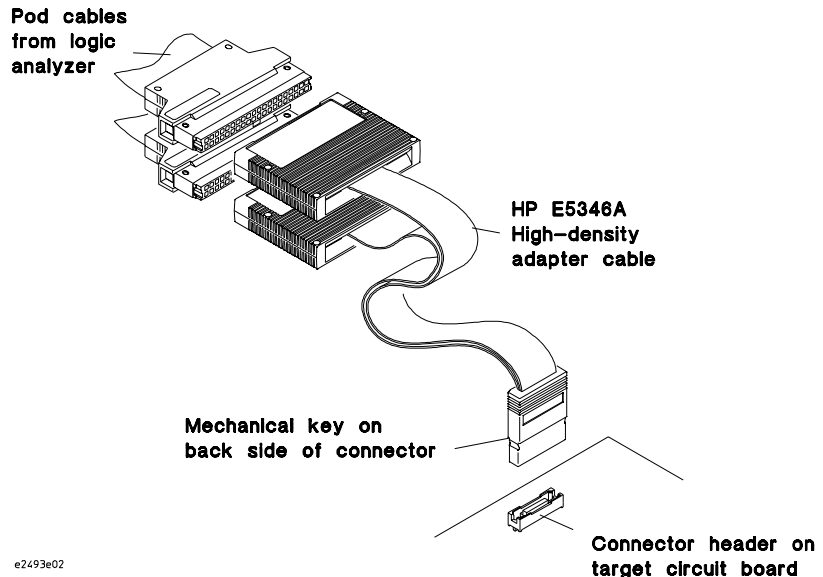
Choosing a Connector Type

logic analyzer using General Purpose probes. Medium-density connector headers can be probed directly, and high-density connector headers can use an adapter assembly for access to the signal pins.

High-Density Connectors

High-density AMP MICTOR connectors are recommended for connecting the target system to the logic analyzer because they require less board space and provide higher signal integrity than medium-density connectors. Each connector carries 32 signals and two clocks.

- Each 32-signal high-density connector header requires approximately 1.1" x 0.4" of PC board space.
- On-board termination is not required.
- Each MICTOR connector requires one Agilent Technologies E5346A High-Density Termination Adapter cable to attach to the logic analyzer. This is a Y-cable where the single end connects to the high-density connector header and each of the two opposite ends connects to a logic analyzer pod.



- Any probed signal line must be able to supply a minimum of 600 mV to the probe tip and handle a minimum loading of 90 kOhms shunted by 10 pF. The maximum input voltage for the logic analyzer is +/- 40 volts peak.

Chapter 2: Preparing the Target System for ARM Core and AMBA ASB

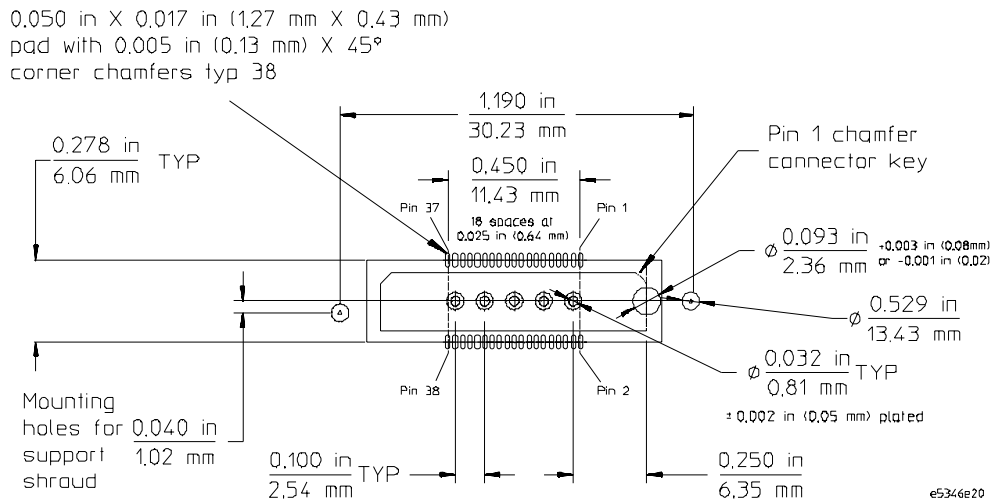
High-Density Connectors

- A plastic shroud (Agilent part number E5346-44701) is available to secure the mechanical connection of the high-density cable to the MICTOR connector header.

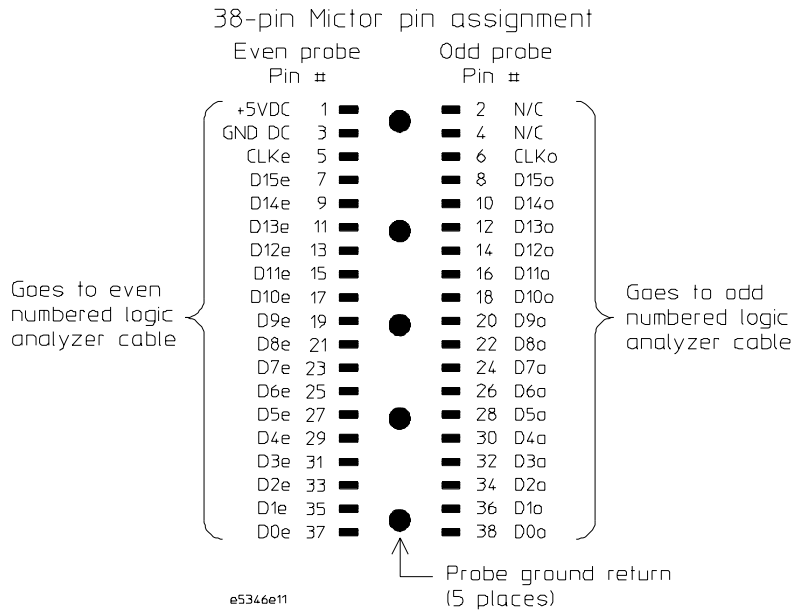
NOTE:

If a PC board already has a high-density connector header attached, but the signal pinouts do not match the requirement, use the MICTOR adapter assembly (Agilent part number E2476-61604) to gain access to the target board signals. Then, use the General Purpose probes from the logic analyzer to connect to the adapter assembly according to the tables later in this Chapter.

Dimensions of the AMP MICTOR 2-767004-2 surface mount connector are shown below. The holes for mounting a support shroud are off-center to allow 0.40 in (1.20 mm) centers when using multiple connectors.



The logic analyzer signal pinouts of the high-density connector are:



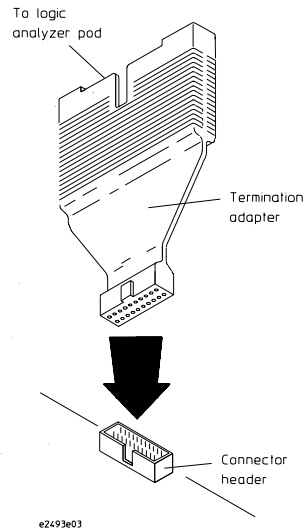
Signal mapping tables listing the ARM core and AMBA ASB signals that are to be routed to each pin of the high-density connector headers are located later in this chapter.

More information on this connector is available by visiting our website at **www.agilent.com** and searching for “E5346A” or “High Density Termination Adapter.”

Medium-Density Connectors

Medium-density connectors carry 16 signals plus one clock. These connectors are an older technology and are not recommended for system clock speeds above 50 MHz.

- Each 16-signal medium-density connector header requires approximately 1.4" x 0.4" of PC board space.
- For each board connector, a 100 kOhm termination adapter (Agilent Part 01650-63203) is required. The termination adapter connects between the connector header and the logic analyzer pod.
- On-board termination is not required.
- Connector pins are spaced on 0.10" centers
- Any probed signal line must be able to supply a minimum of 600 mV to the probe tip and handle a minimum loading of 90 kOhms shunted by 10 pF. The maximum input voltage for the logic analyzer is +/- 40 volts peak.

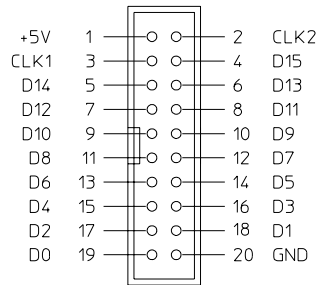


NOTE:

If a PC board already has a connector header attached, but the signal pinouts do not match the requirement, use the General Purpose probes to connect the logic analyzer pods to the connector headers.

Chapter 2: Preparing the Target System for ARM Core and AMBA ASB Medium-Density Connectors

These are the pinouts for the medium-density connector header:



2x10 Header

e2449b02

Application Note 1244-1, “*Minimizing Intrusion Effects When Probing With a Logic Analyzer*” describes this connector in more detail. See *More Information* at the end of this document.

Signal mapping tables listing the ARM core and AMBA ASB signals that are to be routed to each pin of the medium-density connector headers are located later in this chapter.

Signal Requirements for ARM Core or AMBA ASB Inverse Assembly

The Agilent Technologies ARM inverse assembler requires a minimum set of signals for correct operation, others are optional. In general, the presence of optional signals will enhance the capability of the inverse assembler and improve the triggering and storage qualification of the logic analyzer. They may not be present due to cost, pin or space constraints. Each signal used by the inverse assembler is listed and described below. In the case of each optional signal, a discussion describing the impact of its presence or absence is included.

The signals listed are signals coming from the ARM core. For systems using AMBA ASB, the equivalent AMBA ASB signal is listed in {curly braces} after the core signal.

- Required Signals are necessary for basic inverse assembly.
- Strongly Recommended Signals allow the inverse assembler to accurately decode instructions. Without these, some states will be wrong.
- Recommended Signals expand the triggering and storage qualification capability of the logic analyzer, and provide more useful information in the inverse assembler output.
- Optional Signals allow the inverse assembler to differentiate between cycle types and modes.

Required Signals

M_CLK {B_CLK}

MCLK is used to clock data into a logic analyzer (state mode). If MCLK is not available, an alternate clock can be used with the limitations described in the paragraph below. ECLK can be used if the full 32 bit data bus is provided. This will prevent storing wait states and internal cycles which allows more of the trace depth to be used for useful information. With an 8 or 16-bit data bus, the ECLK signal from the core is not usable because it will not clock the extra memory cycles required for a full word access on the reduced data bus. In a similar way, an MCLK stretched by core wait states will not clock the extra memory cycles for narrow memory buses. The use of a stretched MCLK is not recommended when using data buses less than 32 bits.

An alternative logic analysis clock could be generated from MCLK and the memory controller so that it only clocks valid memory cycles, but care must be taken so that the address, data and status signals provided are all valid on the appropriate edges of the clock (data and status signals nMREQ, SEQ and nEXEC on falling edge of the clock; address, chip selects and remaining status on rising edge of the clock).

ADDR[31:0] {B_A[31:0]}

When reduced data buses are used, the address signals A1 and A0 must be the output signals from the memory controller, not from the core. A0 for halfword accesses, and both A0 and A1 for word accesses, are undefined when coming directly off the core and will not correctly inform the inverse assembler which byte(s) is being accessed. Fortunately, the memory controller output is usually what is coming off the chip instead of the core signals since the address signals are required for the memory system interface.

Only the address lines actually used are required for inverse assembly. Upper address lines that are not used do not need to be connected to the logic analyzer. Upper address lines connected to the logic analyzer can be tied low on the headers or left unconnected. (Unconnected inputs to the logic analyzer are pulled low by the logic analyzer).

When using the Agilent 16700-series logic analysis system, chip select signals may be used to regenerate missing upper address lines. This allows the inverse assembler to display the full logical address and allows source referencing to work correctly. (A maximum of 8 different chip selects can be defined.) Source referencing is only supported on the Agilent 16700-series logic analysis system.

Chapter 2: Preparing the Target System for ARM Core and AMBA ASB

Signal Requirements for ARM Core or AMBA ASB Inverse Assembly

If an internal DRAM controller is being used and RAS/CAS signals are coming directly off the chip instead of normal address lines, custom target system hardware must be provided to reconstruct the address bus.

Chip Selects

Chip selects are not generated by the core and are not required. However, if they are being brought off the chip and the upper address lines are not, they can be used by the inverse assembler to regenerate upper address lines.

The inverse assembler supports a maximum of 8 chip selects. The logic analyzer expects the chip select lines to be active low. The lack of upper address lines will make triggering and storage qualification more complicated (they will have to be based on actual chip select and lower address lines instead of logical address used by the programmer).

Either the full set of address lines used or a combination of address lines and chip selects used are required for proper operation of the inverse assembler. If both chip selects and a full set of address lines are available, connect the address lines to the logic analyzer.

DATA[7:0] {B_D[7:0]}

DATA[15:0] {B_D[15:0]}

DATA[31:0] {B_D[31:0]}

The inverse assembler has been designed to operate correctly for systems with reduced data buses. However, reduced data buses will complicate or prevent triggering on data values that are wider than the data bus width because the data value will appear over multiple bus cycles. Reduced data buses will also use more trace depth for capture since each memory cycle will generate multiple states captured by the logic analyzer. To overcome these problems, the full 32-bit data bus can be reconstructed using custom target hardware.

Strongly Recommended Signals

If the following signals are missing, the inverse assembler will make the best attempt possible at decoding the bus cycles correctly, but some states will be wrong. If at all possible, these signals should be provided.

nOPC {B_PROT0}

nOPC is used to distinguish opcode fetches from data accesses. If this signal is missing, the inverse assembler will use the memory map to make this determination. The method is imperfect in cases where data “literal pools” exist in a region that otherwise contains only code. Unexecuted prefetches will not be marked if this signal is not available. For this reason, nOPC is strongly recommended.

MAS[1] {SIZE[1]}

MAS[1] is used to distinguish THUMB fetches from ARM fetches and 32-bit data accesses from smaller data accesses. If this signal is missing, the inverse assembler will use the memory map to make this determination. The memory map requires that the user’s system have distinct memory regions that are exclusively ARM or exclusively THUMB. The map can support a maximum of eight distinct regions. For systems that do a lot of switching between ARM and THUMB instructions, the MAS[1] signal should be considered a required signal.

DMA {A_GNTx (an OR of all A_GNTx other than the processor)}

The ARM core does not directly support DMA. Direct memory accesses require a bus arbiter external to the core. The AMBA ASB specification provides for a bus arbiter. To be consistent with AMBA ASB, the DMA signal should be active high (high when any DMA device has the bus, low when core has the bus) and valid at the rising edge of MCLK. The inverse assembler will use this signal to distinguish DMA memory accesses from memory accesses initiated by the ARM core. These states will be marked as such and not decoded as opcodes or core memory accesses. If DMA is used and the cycles are visible to the logic analyzer, this signal is required, otherwise DMA cycles will be decoded as processor cycles and will confuse the algorithms used by the inverse assembler.

This signal will allow these states to be store-qualified by the logic analyzer or filtered out of the trace listing display if desired.

Recommended Signals

The following signals are useful to allow the inverse assembler to provide more information to the user. If these signals are missing, the inverse assembler will mark states correctly, but some information that could be provided to the user will be missing.

In addition to their usefulness to the inverse assembler, these recommended signals are often useful for storage qualification or triggering of the logic analyzer.

MAS[0] {SIZE[0]}

MAS[0] is used along with MAS[1] for determining the size of data accesses. The inverse assembler will attempt to determine size for data accesses by matching up the data access with the instruction which initiated it. This is not possible in all cases, particularly when storage qualification is being used. The logic analyzer will be unable to trigger or storage qualification based on the data access size if the MAS signals are not available.

nRW {B_WRITE}

The nRW signal is used to distinguish reads from writes. The inverse assembler will attempt to determine read vs. write for data accesses by matching up the data access with the instruction which initiated it. This is not possible in all cases, particularly when store-qualify is being used. Other ASIC specific read/write signals may be available on the target system which can be used for trigger or store-qualify by the logic analyzer but will not be used by the inverse assembler.

nEXEC {INSTEEXEC (ARM9 only)}

The nEXEC signal is used to determine whether conditional instructions were executed. For systems using virtually all THUMB instructions, this signal is not very useful since the only conditional THUMB instructions are branches which can be detected using a different mechanism. For systems with lots of ARM code, this is a very useful signal since most instructions can be conditional. There is no ARM7 AMBA ASB equivalent, you must use the core signal. The ARM9 AMBA ASB equivalent is INSTEEXEC.

DBGACK {no AMBA ASB equivalent, must use core signal}

The DBGACK signal indicates if the processor is running user code or in

Chapter 2: Preparing the Target System for ARM Core and AMBA ASB **Signal Requirements for ARM Core or AMBA ASB Inverse Assembly**

debug mode. This signal is not needed for correct inverse assembly though it is used to mark debug states. However, it is extremely useful for triggering and storage qualification so that only user code execution is captured by the logic analyzer. This signal will also be used for run control, if available.

Optional Signals

Like the recommended signals, the following optional signals allow the inverse assembler to provide more information to the user. If these signals are missing, the inverse assembler will mark states correctly, but some information that could be provided to the user will be missing.

nWAIT, nMREQ, SEQ {B_WAIT B_TRAN1 B_TRAN0}

These signals are used to determine unexecuted prefetches due to branches. Some prefetches can be detected without these signals, but these signals are required to detect all possible unexecuted prefetches.

nMREQ and nWAIT are used to distinguish between memory wait states and internal cycles. Without nMREQ and nWAIT, internal cycles are marked as wait states.

NOTE:

For AMBA ASB systems that use an ARM9 core, the signals are ignored.

nTRANS {B_PROT1}

The nTRANS signal indicates whether the ARM processor is in *user* or *supervisor* mode. If nTRANS is provided, all *supervisor* states will be marked.

ABORT {B_ERROR}

The ABORT signal indicates that the current memory access is not allowed. If this signal is included, then any ABORT states will be marked.

(no core equivalent) {B_LAST}

This signal is only useful with the AMBA ASB inverse assembler. If this signal is available, then bus retracts will be seen.

BIGEND {no AMBA ASB equivalent}

This signal is only useful in systems that switch between big endian and little endian on the fly (for example to access different peripherals that require different endian modes). The inverse assembler will allow the user to select between “Big Endian”, “Little Endian” or “Use BIGEND signal”. For the vast majority of systems, this signal is not needed.

ARM7/ARM9 Core Signal Details

The following table lists several ARM7/ARM9 core signals and their meanings. See pages 45, 46, and 48 for definitions of strongly recommended, recommended, and optional signals.

Signal Name	Importance	Description	Details
MCLK	Required	Clock used for sampling.	clock
nOPC	Strongly Recommended	Used to distinguish instructions from data.	0=instruction 1=data
MAS [1:0]	Strongly Recommended	Indicates memory access size.	00=8-bit 01=16-bit 10=32-bit 11=undefined
DMA	Strongly Recommended	Signals a DMA transfer.	1=DMA transfer
nRW	Recommended	Distinguishes reads from writes.	0=read 1=write
nEXEC/INSTEEXEC	Recommended	Signals whether a conditional instruction was executed. This signal is valid for the instruction that was fetched two states prior.	ARM7: 0=executed ARM9: 1=executed
DBGACK	Recommended	Indicates a debug cycle.	0=normal 1=debug
nTRANS	Optional	Distinguishes user access from supervisor access.	0=user 1=supervisor
nWAIT	Optional	Specifies a wait state.	0=wait state
nMREQ, SEQ	Optional	Specifies internal cycles. Valid for the next memory cycle.	00=sequential memory read 01=non-sequential memory read 10=internal cycle 11=reserved
BIGEND	Optional	Selects endian mode.	1=big endian 0=little endian
ABORT	Optional	Used to signal memory faults.	0=normal 1=memory fault

ARM7/ARM9 Core to AMBA ASB Signal Conversion

The following table shows ARM7/ARM9 core signals and their AMBA ASB counterparts.

ARM7/ARM9	AMBA ASB
MCLK	B_CLK
nOPC	B_PROT [0]
MAS [1:0]	B_SIZE [1:0]
nRW	B_WRITE
nEXEC	INSEXEC**
DBGACK	no equivalent
nTRANS	B_PROT [1]
nWAIT	B_WAIT*
nMREQ	B_TRAN [1]
SEQ	B_TRAN [0]
BIGEND	no equivalent
ABORT	B_ERROR
DMA	no equivalent

*Inverted version of the core signal.

** ARM9 only. No ARM7 AMBA ASB equivalent.

ARM7 AMBA ASB and ARM9 inverse assembler status pod signals

	STAT Pod Bit																							
	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
B_TRAN[0]																								
B_TRAN[0]																								
INSTEXEC																								
B_WRITE																								
B_PROT[0]																								
B_PROT[1]																								
B_SIZE[1:0]																								
DBGACK																								
B_WAIT																								
B_ERROR																								
B_LAST																								
DMA																								
BIGEND																								
CS [7:0]																								

Additional Considerations

1. In order for the inverse assembler to properly read the incoming status signals, the ordering of the status signals must not be modified and the placement of the signals within the STAT label must not be modified.

Example

To have the inverse assembler use the chip select signals to regenerate upper address bits, all status bits to the right of the chip selects must have a placeholder in the STAT label.

If one or more of the status signals are not present in the target system, then the user can safely tie the signal high or low, or simply make them “unconnected” in the Signals dialog (see page 158).

2. SEQ{B_TRAN[0]}, nMREQ{B_TRAN[1]}, and nEXEC{INSTEXEC for ARM9 only} should be master clocked (with the falling edge of MCLK{B_CLK}).

Chapter 2: Preparing the Target System for ARM Core and AMBA ASB
Signal Requirements for ARM Core or AMBA ASB Inverse Assembly

3. All other status signals should be slave clocked (with the rising edge of MCLK{B_CLK}).
4. If a status bit is not connected, there still must be a placeholder for it in the STAT label.
5. CS [7:0] are required status bits if present on the target system.
6. Optional status bits are BIGEND, DMA, and ABORT, {B_ERROR}.

Signal-to-Connector Mappings - ARM Core

The tables on the following pages show the signal-to-connector mappings required by the inverse assembler.

When you have chosen a target-to-logic analyzer connector type, locate the tables on the following pages that apply to your connector type and ARM7 system type.

For ARM systems with an 8-bit data bus:

- Two high-density connector headers are required -or-
- Four medium-density connector headers are required.

For ARM systems with a 16-bit data bus, no chip selects and a 24-bit or smaller data bus:

- Two high-density connector headers are required -or-
- Four medium-density connector headers are required

For ARM systems with a 16-bit or 32-bit data bus:

- Three high-density connector headers are required -or-
- Six medium-density connector headers are required.

The User Defined pins can be used for any signals that you want to route to the logic analyzer.

ARM core signal connector mappings for 8-bit bus systems

The following tables list the signal mappings from an ARM 8-bit bus system to medium-density and high-density connectors. The logic analyzer bit used is listed as a reference. Use these tables to route ARM signals to the connector type you have chosen.

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C1	Hi-Density Connector C1-Odd
NC		1	2
NC		2	4
MCLK	Clk	3	6
User Defined	15	4	8
User Defined	14	5	10
User Defined	13	6	12
User Defined	12	7	14
User Defined	11	8	16
nEXEC	10	9	18
nMREQ	9	10	20
SEQ	8	11	22
DATA 7	7	12	24
DATA 6	6	13	26
DATA 5	5	14	28
DATA 4	4	15	30
DATA 3	3	16	32
DATA 2	2	17	34
DATA 1	1	18	36
DATA 0	0	19	38
GND	Gnd	20	Center

ARM core signal connector mappings for 8-bit bus systems (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C2	Hi-Density Connector C1-Even
NC		1	1
NC		2	3
CS6	Clk	3	5
CS5	15	4	7
CS4	14	5	9
CS3	13	6	11
CS2	12	7	13
CS1	11	8	15
CS0	10	9	17
BIGEND	9	10	19
DMA	8	11	21
ABORT	7	12	23
nWAIT	6	13	25
DBGACK	5	14	27
MAS[1]	4	15	29
MAS[0]	3	16	31
nTRANS	2	17	33
nOPC	1	18	35
nRW	0	19	37
GND	Gnd	20	center

ARM core signal connector mappings for 8-bit bus systems (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C3	Hi-Density Connector C2-Odd
NC		1	2
NC		2	4
CS7	Clk	3	6
ADDR 15	15	4	8
ADDR 14	14	5	10
ADDR 13	13	6	12
ADDR 12	12	7	14
ADDR 11	11	8	16
ADDR 10	10	9	18
ADDR 9	9	10	20
ADDR 8	8	11	22
ADDR 7	7	12	24
ADDR 6	6	13	26
ADDR 5	5	14	28
ADDR 4	4	15	30
ADDR 3	3	16	32
ADDR 2	2	17	34
ADDR 1	1	18	36
ADDR 0	0	19	38
GND	Gnd	20	Center

ARM core signal connector mappings for 8-bit bus systems (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C4	Hi-Density Connector C2-Even
NC		1	1
NC		2	3
User Defined	Clk	3	5
ADDR 31	15	4	7
ADDR 30	14	5	9
ADDR 29	13	6	11
ADDR 28	12	7	13
ADDR 27	11	8	15
ADDR 26	10	9	17
ADDR 25	9	10	19
ADDR 24	8	11	21
ADDR 23	7	12	23
ADDR 22	6	13	25
ADDR 21	5	14	27
ADDR 20	4	15	29
ADDR 19	3	16	31
ADDR 18	2	17	33
ADDR 17	1	18	35
ADDR 16	0	19	37
GND	Gnd	20	Center

ARM core signal connector mappings for 16-bit bus systems (reduced address mode)

The three tables below list the signal mappings from an ARM 16-bit bus system to medium-density and high-density connectors for reduced address mode (24-bit address/16-bit data). The logic analyzer bit used is listed as a reference. Use these tables to route ARM signals to the connector type you have chosen.

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C1	Hi-Density Connector C1-Odd
NC		1	2
NC		2	4
MCLK	Clk	3	6
User Defined	15	4	8
User Defined	14	5	10
User Defined	13	6	12
User Defined	12	7	14
User Defined	11	8	16
nEXEC	10	9	18
nMREQ	9	10	20
SEQ	8	11	22
DATA 7	7	12	24
DATA 6	6	13	26
DATA 5	5	14	28
DATA 4	4	15	30
DATA 3	3	16	32
DATA 2	2	17	34
DATA 1	1	18	36
DATA 0	0	19	38
GND	Gnd	20	Center

ARM core signal connector mappings for 16-bit bus systems (reduced address mode) (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C2	Hi-Density Connector C1-Even
NC		1	1
NC		2	3
ADDR 22	Clk	3	5
ADDR 5	15	4	7
ADDR 4	14	5	9
ADDR 3	13	6	11
ADDR 2	12	7	13
ADDR 1	11	8	15
ADDR 0	10	9	17
BIGEND	9	10	19
DMA	8	11	21
ABORT	7	12	23
nWAIT	6	13	25
DBGACK	5	14	27
MAS[1]	4	15	29
MAS[0]	3	16	31
nTRANS	2	17	33
nOPC	1	18	35
nRW	0	19	37
GND	Gnd	20	center

ARM core signal connector mappings for 16-bit bus systems (reduced address mode) (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C3	Hi-Density Connector C2-Odd
NC		1	2
NC		2	4
ADDR 23	Clk	3	6
ADDR 21	15	4	8
ADDR 20	14	5	10
ADDR 19	13	6	12
ADDR 18	12	7	14
ADDR 17	11	8	16
ADDR 16	10	9	18
ADDR 15	9	10	20
ADDR 14	8	11	22
ADDR 13	7	12	24
ADDR 12	6	13	26
ADDR 11	5	14	28
ADDR 10	4	15	30
ADDR 9	3	16	32
ADDR 8	2	17	34
ADDR 7	1	18	36
ADDR 6	0	19	38
GND	Gnd	20	Center

ARM core signal connector mappings for 16-bit bus systems (reduced address mode) (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C472	Hi-Density Connector C2-Even
NC		1	1
NC		2	3
User Defined	Clk	3	5
User Defined	15	4	7
User Defined	14	5	9
User Defined	13	6	11
User Defined	12	7	13
User Defined	11	8	15
User Defined	10	9	17
User Defined	9	10	19
User Defined	8	11	21
DATA 15	7	12	23
DATA 14	6	13	25
DATA 13	5	14	27
DATA 12	4	15	29
DATA 11	3	16	31
DATA 10	2	17	33
DATA 9	1	18	35
DATA 8	0	19	37
GND	Gnd	20	Center

ARM core signal mappings for 16/32-bit bus systems

The six tables below list the signal mappings from an ARM 16-bit or 32-bit bus system to medium-density and high-density connectors. The logic analyzer bit used is listed as a reference. Use these tables to route ARM signals to the connector type you have chosen

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C1	Hi-Density Connector C1-Odd
NC		1	2
NC		2	4
MCLK	Clk	3	6
DATA 15	15	4	8
DATA 14	14	5	10
DATA 13	13	6	12
DATA 12	12	7	14
DATA 11	11	8	16
DATA 10	10	9	18
DATA 9	9	10	20
DATA 8	8	11	22
DATA 7	7	12	24
DATA 6	6	13	26
DATA 5	5	14	28
DATA 4	4	15	30
DATA 3	3	16	32
DATA 2	2	17	34
DATA 1	1	18	36
DATA 0	0	19	38
GND	Gnd	20	Center

ARM core signal connector mappings for 16/32-bit bus systems (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C2	Hi-Density Connector C1-Even
NC		1	1
NC		2	3
User Defined	Clk	3	5
DATA 31	15	4	7
DATA 30	14	5	9
DATA 29	13	6	11
DATA 28	12	7	13
DATA 27	11	8	15
DATA 26	10	9	17
DATA 25	9	10	19
DATA 24	8	11	21
DATA 23	7	12	23
DATA 22	6	13	25
DATA 21	5	14	27
DATA 20	4	15	29
DATA 19	3	16	31
DATA 18	2	17	33
DATA 17	1	18	35
DATA 16	0	19	37
GND	Gnd	20	Center

ARM core signal connector mappings for 16/32-bit bus systems (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C3	Hi-Density Connector C2-Odd
NC		1	2
NC		2	4
User Defined	Clk	3	6
User Defined	15	4	8
User Defined	14	5	10
User Defined	13	6	12
User Defined	12	7	14
User Defined	11	8	16
User Defined	10	9	18
User Defined	9	10	20
User Defined	8	11	22
User Defined	7	12	24
User Defined	6	13	26
User Defined	5	14	28
User Defined	4	15	30
User Defined	3	16	32
nEXEC	2	17	34
nMREQ	1	18	36
SEQ	0	19	38
GND	Gnd	20	Center

ARM core signal connector mappings for 16/32-bit bus systems (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C4	Hi-Density Connector C2-Even
NC		1	1
NC		2	3
CS6	Clk	3	5
CS5	15	4	7
CS4	14	5	9
CS3	13	6	11
CS2	12	7	13
CS1	11	8	15
CS0	10	9	17
BIGEND	9	10	19
DMA	8	11	21
ABORT	7	12	23
nWAIT	6	13	25
DBGACK	5	14	27
MAS[1]	4	15	29
MAS[0]	3	16	31
nTRANS	2	17	33
nOPC	1	18	35
nRW	0	19	37
GND	Gnd	20	center

ARM core signal connector mappings for 16/32-bit bus systems (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C5	Hi-Density Connector C3-Odd
NC		1	2
NC		2	4
CS7	Clk	3	6
ADDR 15	15	4	8
ADDR 14	14	5	10
ADDR 13	13	6	12
ADDR 12	12	7	14
ADDR 11	11	8	16
ADDR 10	10	9	18
ADDR 9	9	10	20
ADDR 8	8	11	22
ADDR 7	7	12	24
ADDR 6	6	13	26
ADDR 5	5	14	28
ADDR 4	4	15	30
ADDR 3	3	16	32
ADDR 2	2	17	34
ADDR 1	1	18	36
ADDR 0	0	19	38
GND	Gnd	20	Center

ARM core signal connector mappings for 16/32-bit bus systems (continued)

ARM Core Signal	Logic Analyzer Bit	Med-Density Connector C6	Hi-Density Connector C3-Even
NC		1	1
NC		2	3
User Defined	Clk	3	5
ADDR 31	15	4	7
ADDR 30	14	5	9
ADDR 29	13	6	11
ADDR 28	12	7	13
ADDR 27	11	8	15
ADDR 26	10	9	17
ADDR 25	9	10	19
ADDR 24	8	11	21
ADDR 23	7	12	23
ADDR 22	6	13	25
ADDR 21	5	14	27
ADDR 20	4	15	29
ADDR 19	3	16	31
ADDR 18	2	17	33
ADDR 17	1	18	35
ADDR 16	0	19	37
GND	Gnd	20	Center

Signal-to-Connector Mappings - AMBA ASB

The tables on the following pages show the signal-to-connector mappings required by the inverse assembler.

When you have chosen a target-to-logic analyzer connector type, locate the tables on the following pages that apply to your connector type and AMBA ASB system type.

For AMBA ASB systems with an 8-bit data bus:

- Two high-density connector headers are required -or-
- Four medium-density connector headers are required

For AMBA ASB systems with a 16-bit or 32-bit data bus:

- Three high-density connector headers are required -or-
- Six medium-density connector headers are required

The User Defined pins can be used for any signals that you want to route to the logic analyzer.

AMBA ASB signal connector mappings for 8-bit bus systems

The following tables list the signal mappings from an AMBA ASB 8-bit bus system to medium-density and high-density connectors. The logic analyzer bit used is listed as a reference. Use these tables to route AMBA ASB signals to the connector type you have chosen.

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C1	Hi-Density Connector C1-Odd
NC		1	2
NC		2	4
B_CLK	Clk	3	6
User Defined	15	4	8
User Defined	14	5	10
User Defined	13	6	12
User Defined	12	7	14
User Defined	11	8	16
nEXEC (ARM7)	10	9	18
INSTREXEC (ARM9)	10	9	18
B_TRAN[1]	9	10	20
B_TRAN[0]	8	11	22
B_D 7	7	12	24
B_D 6	6	13	26
B_D 5	5	14	28
B_D 4	4	15	30
B_D 3	3	16	32
B_D 2	2	17	34
B_D 1	1	18	36
B_D 0	0	19	38
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 8-bit bus systems (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C2	Hi-Density Connector C1-Even
NC		1	1
NC		2	3
CS5	Clk	3	5
CS4	15	4	7
CS3	14	5	9
CS2	13	6	11
CS1	12	7	13
CS0	11	8	15
BIGEND	10	9	17
DMA	9	10	19
B_LAST	8	11	21
B_ERROR	7	12	23
B_WAIT	6	13	25
DBGACK	5	14	27
SIZE[1]	4	15	29
SIZE[0]	3	16	31
B_PROT[1]	2	17	33
B_PROT[0]	1	18	35
B_WRITE	0	19	37
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 8-bit bus systems (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C3	Hi-Density Connector C2-Odd
NC		1	2
NC		2	4
CS6	Clk	3	6
B_A 15	15	4	8
B_A 14	14	5	10
B_A 13	13	6	12
B_A 12	12	7	14
B_A 11	11	8	16
B_A 10	10	9	18
B_A 9	9	10	20
B_A 8	8	11	22
B_A 7	7	12	24
B_A 6	6	13	26
B_A 5	5	14	28
B_A 4	4	15	30
B_A 3	3	16	32
B_A 2	2	17	34
B_A 1	1	18	36
B_A 0	0	19	38
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 8-bit bus systems (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C4	Hi-Density Connector C2-Even
NC		1	1
NC		2	3
CS7	Clk	3	5
B_A 31	15	4	7
B_A 30	14	5	9
B_A 29	13	6	11
B_A 28	12	7	13
B_A 27	11	8	15
B_A 26	10	9	17
B_A 25	9	10	19
B_A 24	8	11	21
B_A 23	7	12	23
B_A 22	6	13	25
B_A 21	5	14	27
B_A 20	4	15	29
B_A 19	3	16	31
B_A 18	2	17	33
B_A 17	1	18	35
B_A 16	0	19	37
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16-bit bus systems (reduced address mode)

The following tables list the signal mappings from an AMBA ASB 16-bit bus system to medium-density and high-density connectors for reduced address mode (24-bit address/16-bit data). The logic analyzer bit used is listed as a reference. Use these tables to route AMBA ASB signals to the connector type you have chosen.

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C1	Hi-Density Connector C1-Odd
NC		1	2
NC		2	4
B_CLK	Clk	3	6
User Defined	15	4	8
User Defined	14	5	10
User Defined	13	6	12
User Defined	12	7	14
User Defined	11	8	16
nEXEC (ARM7)	10	9	18
INSTREXEC (ARM9)	10	9	18
B_TRAN[1]	9	10	20
B_TRAN[0]	8	11	22
B_D 7	7	12	24
B_D 6	6	13	26
B_D 5	5	14	28
B_D 4	4	15	30
B_D 3	3	16	32
B_D 2	2	17	34
B_D 1	1	18	36
B_D 0	0	19	38
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16-bit bus systems (reduced address mode) (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C2	Hi-Density Connector C1-Even
NC		1	1
NC		2	3
B_A 22	Clk	3	5
B_A 5	15	4	7
B_A 4	14	5	9
B_A 3	13	6	11
B_A 2	12	7	13
B_A 1	11	8	15
B_A 0	10	9	17
DMA	9	10	19
B_LAST	8	11	21
B_ERROR	7	12	23
B_WAIT	6	13	25
DBGACK	5	14	27
SIZE[1]	4	15	29
SIZE[0]	3	16	31
B_PROT[1]	2	17	33
B_PROT[0]	1	18	35
B_WRITE	0	19	37
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16-bit bus systems (reduced address mode) (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C3	Hi-Density Connector C2-Odd
NC		1	2
NC		2	4
B_A 23	Clk	3	6
B_A 21	15	4	8
B_A 20	14	5	10
B_A 19	13	6	12
B_A 18	12	7	14
B_A 17	11	8	16
B_A 16	10	9	18
B_A 15	9	10	20
B_A 14	8	11	22
B_A 13	7	12	24
B_A 12	6	13	26
B_A 11	5	14	28
B_A 10	4	15	30
B_A 9	3	16	32
B_A 8	2	17	34
B_A 7	1	18	36
B_A 6	0	19	38
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16-bit bus systems (reduced address mode) (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C4	Hi-Density Connector C2-Even
NC		1	1
NC		2	3
User Defined	Clk	3	5
User Defined	15	4	7
User Defined	14	5	9
User Defined	13	6	11
User Defined	12	7	13
User Defined	11	8	15
User Defined	10	9	17
User Defined	9	10	19
User Defined	8	11	21
B_D 15	7	12	23
B_D 14	6	13	25
B_D 13	5	14	27
B_D 12	4	15	29
B_D 11	3	16	31
B_D 10	2	17	33
B_D 9	1	18	35
B_D 8	0	19	37
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16/32-bit bus systems

The six tables below list the signal mappings from an AMBA ASB 16-bit or 32-bit bus system to medium-density and high-density connectors. The logic analyzer bit used is listed as a reference. Use these tables to route AMBA ASB signals to the connector type you have chosen

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C1	Hi-Density Connector C1-Odd
NC		1	2
NC		2	4
B_CLK	Clk	3	6
B_D 15	15	4	8
B_D 14	14	5	10
B_D 13	13	6	12
B_D 12	12	7	14
B_D 11	11	8	16
B_D 10	10	9	18
B_D 9	9	10	20
B_D 8	8	11	22
B_D 7	7	12	24
B_D 6	6	13	26
B_D 5	5	14	28
B_D 4	4	15	30
B_D 3	3	16	32
B_D 2	2	17	34
B_D 1	1	18	36
B_D 0	0	19	38
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C2	Hi-Density Connector C1-Even
NC		1	1
NC		2	3
User Defined	Clk	3	5
B_D 31	15	4	7
B_D 30	14	5	9
B_D 29	13	6	11
B_D 28	12	7	13
B_D 27	11	8	15
B_D 26	10	9	17
B_D 25	9	10	19
B_D 24	8	11	21
B_D 23	7	12	23
B_D 22	6	13	25
B_D 21	5	14	27
B_D 20	4	15	29
B_D 19	3	16	31
B_D 18	2	17	33
B_D 17	1	18	35
B_D 16	0	19	37
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C3	Hi-Density Connector C2-Odd
NC		1	2
NC		2	4
User Defined	Clk	3	6
User Defined	15	4	8
User Defined	14	5	10
User Defined	13	6	12
User Defined	12	7	14
User Defined	11	8	16
User Defined	10	9	18
User Defined	9	10	20
User Defined	8	11	22
User Defined	7	12	24
User Defined	6	13	26
User Defined	5	14	28
User Defined	4	15	30
User Defined	3	16	32
nEXEC (ARM7)	2	17	34
INSTREXEC (ARM9)	2	17	34
B_TRAN[1]	1	18	36
B_TRAN[0]	0	19	38
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C4	Hi-Density Connector C2-Even
NC		1	1
NC		2	3
CS5	Clk	3	5
CS4	15	4	7
CS3	14	5	9
CS2	13	6	11
CS1	12	7	13
CS0	11	8	15
BIGEND	10	9	17
DMA	9	10	19
B_LAST	8	11	21
B_ERROR	7	12	23
B_WAIT	6	13	25
DBGACK	5	14	27
SIZE[1]	4	15	29
SIZE[0]	3	16	31
B_PROT[1]	2	17	33
B_PROT[0]	1	18	35
B_WRITE	0	19	37
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C5	Hi-Density Connector C3-Odd
NC		1	2
NC		2	4
CS6	Clk	3	6
B_A 15	15	4	8
B_A 14	14	5	10
B_A 13	13	6	12
B_A 12	12	7	14
B_A 11	11	8	16
B_A 10	10	9	18
B_A 9	9	10	20
B_A 8	8	11	22
B_A 7	7	12	24
B_A 6	6	13	26
B_A 5	5	14	28
B_A 4	4	15	30
B_A 3	3	16	32
B_A 2	2	17	34
B_A 1	1	18	36
B_A 0	0	19	38
GND	Gnd	20	Center

AMBA ASB signal connector mappings for 16/32-bit bus systems (continued)

AMBA ASB Signal	Logic Analyzer Bit	Med-Density Connector C6	Hi-Density Connector C3-Even
NC		1	1
NC		2	3
CS7	Clk	3	5
B_A 31 or CS 7	15	4	7
B_A 30 or CS 6	14	5	9
B_A 29 or CS 5	13	6	11
B_A 28 or CS 4	12	7	13
B_A 27 or CS 3	11	8	15
B_A 26 or CS 2	10	9	17
B_A 25 or CS 1	9	10	19
B_A 24 or CS 0	8	11	21
B_A 23	7	12	23
B_A 22	6	13	25
B_A 21	5	14	27
B_A 20	4	15	29
B_A 19	3	16	31
B_A 18	2	17	33
B_A 17	1	18	35
B_A 16	0	19	37
GND	Gnd	20	Center

Designing a JTAG Connector into Your Target System

For information on designing a JTAG connector into your target system, see the emulation manual supplied with your emulation probe/module.

Preparing the Target System for
AMBA AHB

Chapter 3: Preparing the Target System for AMBA AHB

This chapter describes the necessary connections between the target system and the logic analysis system for inverse assembly on an ARM AMBA AHB target system.

Because ARM circuits will vary with each design, it is important that you design headers into your target system for connection to a logic analyzer.

If your system already contains connector headers with incompatible pinouts, you can still connect to a logic analyzer using an adapter cable and General Purpose probes.

- When using the ARM AMBA AHB inverse assembler, design the target system with high-density AMP MICTOR connectors to connect the target system to the logic analysis system.
- Use Agilent Technologies cables or termination adapters to connect your target system to a logic analyzer.

	High Density Connector
Pin Configuration	2 rows x 19 pins
Header Part Number	AMP 2-767004-2
Required Termination Adapter	Agilent E5346A (one for every two logic analyzer pods)
Notes	Each connector supports two logic analyzer pods

- The Agilent Technologies E5346A is a Y-cable which connects two logic analyzer pods to one high-density connector header while providing the proper termination.

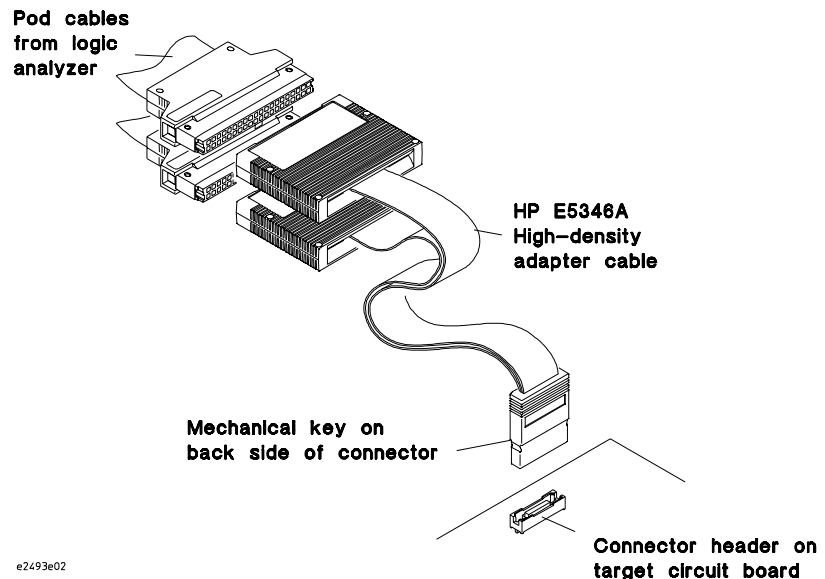
NOTE:

If a PC board already has high-density connector headers attached, but the signal pinouts do not match the requirement, you can still connect to a logic analyzer using General Purpose probes (or you may be able to re-order bits — see your logic analysis system help system for details).

High-Density Connectors

High-density MICTOR connectors are required for connecting the target system to the logic analyzer because they require less board space and provide higher signal integrity than medium-density connectors. Each connector carries 32 signals and two clocks.

- Each 32-signal high-density connector header requires approximately 1.1" x 0.4" of PC board space.
- On-board termination is not required.
- Each MICTOR connector requires one Agilent Technologies E5346A High-Density Termination Adapter cable to attach to the logic analyzer. This is a Y-cable where the single end connects to the high-density connector header and each of the two opposite ends connects to a logic analyzer pod.



- Any probed signal line must be able to supply a minimum of 600 mV to the probe tip and handle a minimum loading of 90 kOhms shunted by 10 pF. The maximum input voltage for the logic analyzer is +/- 40 volts peak.

Chapter 3: Preparing the Target System for AMBA AHB

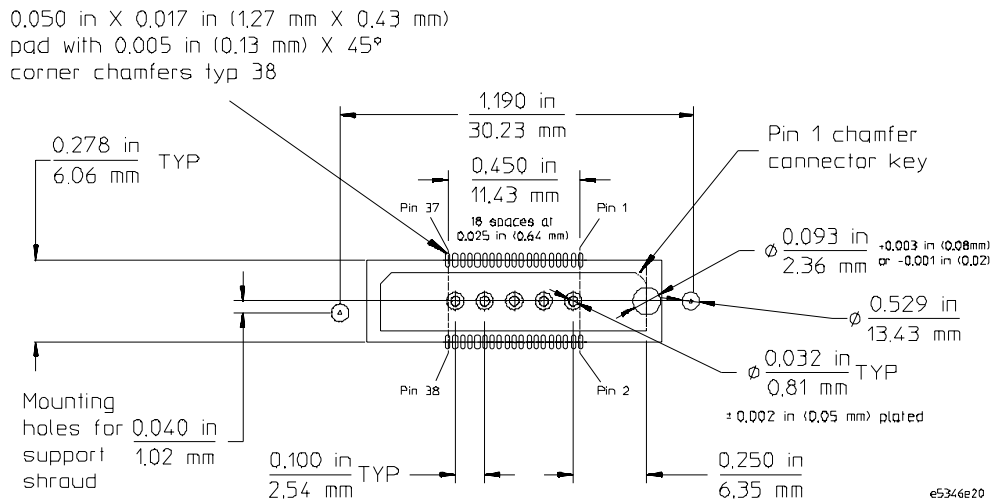
High-Density Connectors

- A plastic shroud (Agilent part number E5346-44701) is available to secure the mechanical connection of the high-density cable to the MICTOR connector header.

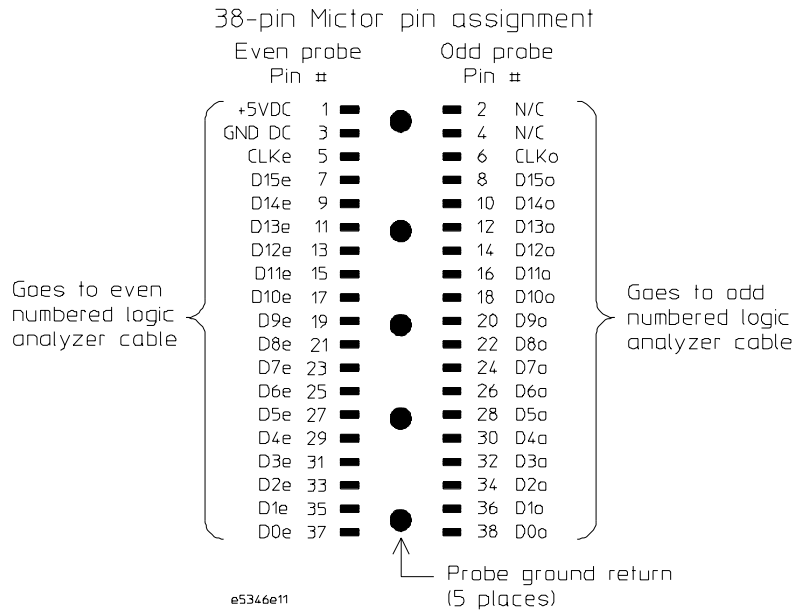
NOTE:

If a PC board already has a high-density connector header attached, but the signal pinouts do not match the requirement, use the MICTOR adapter assembly (Agilent part number E5346-60002) to gain access to the target board signals. Then, use the General Purpose probes from the logic analyzer to connect to the adapter assembly according to the signal connector mapping tables in this Chapter.

Dimensions of the AMP MICTOR 2-767004-2 surface mount connector are shown below. The holes for mounting a support shroud are off-center to allow 0.40 in (1.20 mm) centers when using multiple connectors.



The logic analyzer signal pinouts of the high-density connector are:



Signal mapping tables listing the AMBA AHB signals that are to be routed to each pin of the high-density connector headers are located later in this chapter.

More information on this connector is available by visiting our website at **www.agilent.com** and searching for “E5346A” or “High Density Termination Adapter.”

Signal Requirements for ARM AMBA AHB Inverse Assembly

The Agilent Technologies ARM inverse assembler requires a minimum set of signals for correct operation, others are optional. In general, the presence of optional signals will enhance the capability of the inverse assembler and improve the triggering and storage qualification of the logic analyzer. They may not be present due to cost, pin or space constraints. Each signal used by the inverse assembler is listed and described below. In the case of each optional signal, a discussion describing the impact of its presence or absence is included.

Single Master and Multiple Master Configurations

The inverse assembler supports two connector pinouts: the single master configuration found on the ARM966ES core module board, and the multiple master configuration found on the ARM Integrator/AM board.

Single Master Configuration

Required Signals

Each signal required by the inverse assembler is described below.

HCLK. The rising edge of the HCLK signal is used by the logic analyzer to sample the address, data, and control signals.

HADDR[31:0]. The address lines must be present to capture accurate software flow. The address lines specify which location is being addressed.

HRDATA[31:0]. The HRDATA signals are used by the slave to generate its data. The HRDATA signals can also be used by the master to broadcast write data if logic analyzer connections are limited.

HREADY. The HREADY signal indicates the completion of a transfer. When HREADY is LOW, this signal causes wait states to be inserted into the transfer and allows extra time for the slave to provide or sample data. When HREADY goes HIGH, the transfer has completed.

HTRANS[1:0]. The HTRANS signals are used to classify the four different types of transfers: IDLE, BUSY, NONSEQ, and SEQ.

HWRITE. The HWRITE signal indicates whether the transfer is a read or write. When HWRITE is HIGH, this signal indicates a write transfer and the master will broadcast data on the write data bus, HWDATA[31:0]. When HWRITE is LOW, this signal indicates a read transfer and the slave must generate data on the read data bus, HRDATA[31:0].

HSIZE[1:0]. The HSIZE indicates the size of the transfer. In this configuration, there are three sizes: Byte, Halfword, and Word.

HRESP[1:0]. The HRESP signals indicate the response from the slave to the master. There are four types of responses: OKAY, ERROR, RETRY, and SPLIT.

Note: If the bus is designed for AHB-Lite, the RETRY and SPLIT responses are not used, so HRESP1 is not needed.

HPROTO. The HPROTO signal is used to indicate if the transfer is an opcode fetch or a data access.

Optional Signals

Optional Signals allow the inverse assembler to differentiate between cycle types and modes.

HWDATA[31:0]. The HWDATA signals are used by the master to broadcast its data. This connection is not required if both read and write data are combined on the HRDATA signals. There is a user preferences button to allow the user to inform the inverse assembler if the HWDATA signals are being used for disassembly.

HGRANT. The HGRANT signal is generated by the arbiter and indicates that the appropriate master is currently the highest priority master requesting the bus, taking into account locked and split transfers. When HGRANT is HIGH, the master has control of the bus. When HGRANT is LOW, another master has control of the bus. This signal is used for bus control filtering.

HRESP1. If the bus is designed for AHB-Lite, this signal is not required. There is a user preferences button to allow the user to inform the inverse assembler if the HRESP1 signal is being used for disassembly.

Other Signals Not Used for Inverse Assembly

These signals are part of the AMBA AHB specification but are not used by the inverse assembler.

HBURST[2:0]. The HBURST signals are used to classify the eight different types of burst information: SINGLE, INCR, WRAP4, INCR4, WRAP8, INCR8, WRAP16, and INCR16.

HPROT1. The HPROT1 signal is used to indicate if the transfer is a user access or a privileged access.

HPROT2. The HPROT2 signals is used to indicate if the transfer is not bufferable or bufferable.

HPROT3. The HPROT3 signal is used to indicate if the transfer is not cacheable or cacheable.

HLOCK. The HLOCK signal is asserted at the same time as the bus request signal. This indicates to the arbiter that the master is performing a number of indivisible transfers and the arbiter must not grant any other bus master access to the bus once the first transfer of the locked transfer has

commenced.

HBUSREQ. The HBUSREQ signal is used by the bus master to request access to the bus.

ARM AMBA AHB Signal Details — Single Master

The following table lists some ARM AMBA AHB signals and their meanings. See “Signal Requirements for ARM AMBA AHB Inverse Assembly” beginning on page 90 for more information,

Signal Name	Importance	Description	Details
HCLK	Required	Clock used for sampling.	Clock
HADDR [31:0]	Required	Address signals.	
HRDATA [31:0]	Required	The read data bus and possibly the write data bus.	
HREADY	Required	Signals the completion of a transfer.	0=wait 1=ready
HTRANS [1:0]	Required	Used to classify the type of transfer.	00=idle 01=busy 10=nonseq 11=seq
HWRITE	Required	Indicates if the transfer is a read or write.	0=read 1=write
HSIZE [1:0]	Required	Indicates the size of the transfer.	00=byte 01=halfword 10=word
HRESP [1:0]	Required (Full AHB)	Indicates the response from the slave.	00=okay 01=error 10=retry 11=split
HPROTO	Required	Determines whether the transfer is an opcode fetch or a data access.	0=opcode fetch 1=data access
HWDATA[31:0]	Optional	The write data bus. It is optional if the write data bus information is also routed to the HRDATA bus.	

Chapter 3: Preparing the Target System for AMBA AHB
Single Master Configuration

Signal Name	Importance	Description	Details
HGRANT	Optional	Indicates bus control.	0=another master has control 1=this master has control
HRESP1	Optional (AHB-Lite)	Not required if this is an AHB-Lite system.	
HBURST[2:0]	Not used for inverse assembly	Used to classify the burst type.	000=single 001=incr 010=wrap4 011=incr4 100=wrap8 101=incr8 110=wrap8 111=incr16
HPROT1	Not used for inverse assembly	Indicates the transfer's access type.	0=user access 1=privilege access
HPROT2	Not used for inverse assembly	Indicates if the transfer is bufferable.	0=not bufferable 1=bufferable
HPROT3	Not used for inverse assembly	Indicates if the transfer is cacheable.	0=not cacheable 1=cacheable
HLOCK	Not used for inverse assembly	Indicates if the transfer is locked.	0=not locked 1=locked
HBUSREQ	Not used for inverse assembly	Used by the bus master to request access to the bus	0=no request 1=request access

Multiple Master Configuration

Required Signals

Each signal required by the inverse assembler is described below.

HCLK. The rising edge of the HCLK signal is used by the logic analyzer to sample the address, data, and control signals.

ADDR[31:0]. The address lines must be present to capture accurate software flow. The address lines specify which location is being addressed. The address lines are sampled on the rising edge of the HCLK signal.

DATA[31:0]. The DATA signals are used by the slave to generate its read data and by the master to broadcast write data.

HREADY. The HREADY signal indicates the completion of a transfer. When HREADY is LOW, this signal causes wait states to be inserted into the transfer and allows extra time for the slave to provide or sample data. When HREADY goes HIGH, the transfer has completed.

HTRANS[1:0]. The HTRANS signals are used to classify the four different types of transfers: IDLE, BUSY, NONSEQ, and SEQ.

HWRITE. The HWRITE signal indicates whether the transfer is a read or write. When HWRITE is HIGH, this signal indicates a write transfer and the master will broadcast data. When HWRITE is LOW, this signal indicates a read transfer and the slave must generate data.

HSIZE[1:0]. The HSIZE indicates the size of the transfer. In this configuration, there are three sizes: Byte, Halfword, and Word.

HRESP[1:0]. The HRESP signals indicate the response from the slave to the master. There are four types of responses: OKAY, ERROR, RETRY, and SPLIT.

NOTE:

If the bus is designed for AHB-Lite, the RETRY and SPLIT responses are not used, so HRESP1 is not needed.

HPROTO. The HPROTO signal is used to indicate if the transfer is an opcode fetch or a data access.

Optional AHB Signals

HGRANT[3:0]. The HGRANT signal is generated by the arbiter and indicates that the appropriate master is currently the highest priority master requesting the bus, taking into account locked and split transfers. When HGRANTx is HIGH, the master has control of the bus. When HGRANTx is LOW, another master has control of the bus. These signals are used for bus control filtering.

HRESP1. If the bus is designed for AHB-Lite, this signal is not required. There is a user preferences button to allow the user to inform the inverse assembler if the HRESP1 signal is being used for disassembly.

Other AHB Signals Not Used for Inverse Assembly

These signals are part of the AMBA AHB specification but are not used by the inverse assembler.

HBURST[2:0]. The HBURST signals are used to classify the eight different types of burst information: SINGLE, INCR, WRAP4, INCR4, WRAP8, INCR8, WRAP16, and INCR16.

HPROT1. The HPROT1 signal is used to indicate if the transfer is a user access or a privileged access.

HPROT2. The HPROT2 signals is used to indicate if the transfer is not bufferable or bufferable.

HPROT3. The HPROT3 signal is used to indicate if the transfer is not cacheable or cacheable.

HLOCK[3:0]. The HLOCK signals is asserted at the same time as the bus request signal. This indicates to the arbiter that the master is performing a number of indivisible transfers and the arbiter must not grant any other bus master access to the bus once the first transfer of the locked transfer has commenced.

HBUSREQ[3:0]. The HBUSREQ signals is used by the bus master to request access to the bus.

MASTER[2:0]. The arbiter indicates which master is currently granted the

bus using the MASTER signals and this can be used to control the central address and control multiplexor. The master number is also required by SPLIT-capable slaves that they can indicate to the arbiter which master is able to complete a SPLIT transfer.

HMASTLOCK. The arbiter indicates that the current transfer is part of a locked sequence by asserting the HMASTLOCK signal, which has the same timing as the address and control signals.

HSPLIT[5:0]. The Split Complete bus is used by a SPLIT-capable slave to indicate which bus master can complete a SPLIT transaction. This information is needed by the arbiter so that it can grant the master access to the bus to complete the transfer.

ARM AMBA AHB Signal Details — Multiple Masters

The following table lists some ARM AMBA AHB signals and their meanings. See “Signal Requirements for ARM AMBA AHB Inverse Assembly” beginning on page 90 for more information,

Signal Name	Importance	Description	Details
HCLK	Required	Clock used for sampling.	Clock
ADDR[31:0]	Required	Address signals.	
DATA[31:0]	Required	The read data bus and possibly the write data bus.	
HREADY	Required	Signals the completion of a transfer.	0=wait 1=ready
HTRANS[1:0]	Required	Used to classify the type of transfer.	00=idle 01=busy 10=nonseq 11=seq
HWRITE	Required	Indicates if the transfer is a read or write.	0=read 1=write
HSIZE[1:0]	Required	Indicates the size of the transfer.	00=byte 01=halfword 10=word

Chapter 3: Preparing the Target System for AMBA AHB
Multiple Master Configuration

Signal Name	Importance	Description	Details
HRESP[1:0]	Required (Full AHB)	Indicates the response from the slave.	00=okay 01=error 10=retry 11=split
HPROTO	Required	Determines if the transfer is an opcode fetch or a data access.	0=opcode fetch 1=data access
HGRANT[3:0]	Optional	Indicates bus control.	A high indicates which master has control.
HRESP1	Optional (AHB-Lite)	Not required if this is an AHB-Lite system.	
HBURST[2:0]	Not used for inverse assembly	Used to classify the burst type.	000=single 001=incr 010=wrap4 011=incr4 100=wrap8 101=incr8 110=wrap8 111=incr16
HPROT1	Not used for inverse assembly	Indicates the transfer's access type.	0=user access 1=privilege access
HPROT2	Not used for inverse assembly	Indicates if the transfer is bufferable.	0=not bufferable 1=bufferable
HPROT3	Not used for inverse assembly	Indicates if the transfer is cacheable.	0=not cacheable 1=cacheable
HLOCK[3:0]	Not used for inverse assembly	Indicates if the transfer is locked.	A high indicates the master generating the a locked transfer.
HBUSREQ[3:0]	Not used for inverse assembly	Used by the bus master to request access to the bus	A high indicates the master that is requesting the bus.

Signal Name	Importance	Description	Details
MASTER[2:0]	Not used for inverse assembly	Indicates the number of the master that has control of the bus	
HMASTLOCK	Not used for inverse assembly	Indicates if the current transfer is part of a locked sequence	0=not locked 1=locked
HSPLIT[5:0]	Not used for inverse assembly	Indicates which bus can complete a split transaction.	A high indicates the master can complete a split transaction.

The following tables indicate the ARM AMBA AHB STAT pod order for multiple master configuration.

		Odd STAT Pod Bit																				
		20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HTRANS [1:0]																						
HBURST [2:0]																						
HSPLIT [5:0]																						

		Even STAT Pod Bit																				
		20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HWRITE																						
HPROTO																						
HPROT1																						
HSIZE [1:0]																						
HMASTLOCK																						
HREADY																						
HRESP [1:0]																						
HPROT2																						
HPROT3																						

		Odd HWDATA Pod Bit																				
		20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HBUSREQ [3:0]																						
HLOCK [3:0]																						

		Even HWDATA Pod Bit																				
		20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HGRANT [3:0]																						
MASTER [2:0]																						

Signal-to-Connector Mappings - AMBA AHB

The tables on the following pages show the signal-to-connector mappings required by the inverse assembler. Connector mapping tables are given for single master and multiple master configurations.

Four high-density connector headers are required when both HRDATA and HWDATA buses are pinned out. When only the HRDATA bus is pinned out, three connectors are needed.

The unused pins can be used for any signals that you want to route to the logic analyzer.

Signal-to-Connector Mappings — Single Master Configuration

The tables below list the signal mappings to high-density connectors for an ARM AMBA AHB system in single master configuration. The logic analyzer bit used is listed as a reference. Use these tables to route ARM signals to the connector.

NOTE:

The HRDATA bus can be used for both the reads and writes if the HWDATA bus is not pinned out. To set the preference accordingly, see “HWDATA” on page 161.

HREADY and HTRANS1 appear twice in the connector pinout definition. The inverse assembler uses the STAT connector pin 25 for HREADY and the STAT connector pin 36 for HTRANS1.

ADDR Signals - Single Master Configuration

LA Connector C1 Even LA Pod			LA Connector C1 Odd LA Pod		
ARM AMBA AHB Signal Name	LA Bit	MICTOR Connector Pin	MICTOR Connector Pin	LA Bit	ARM AMBA AHB Signal Name
no connect		1	2		no connect
no connect		3	4		no connect
HTRANS1	K clk	5	6	J clk	HCLK
HADDR31	15	7	8	15	HADDR15
HADDR30	14	9	10	14	HADDR14
HADDR29	13	11	12	13	HADDR13
HADDR28	12	13	14	12	HADDR12
HADDR27	11	15	16	11	HADDR11
HADDR26	10	17	18	10	HADDR10
HADDR25	9	19	20	9	HADDR9
HADDR24	8	21	22	8	HADDR8
HADDR23	7	23	24	7	HADDR7
HADDR22	6	25	26	6	HADDR6
HADDR21	5	27	28	5	HADDR5
HADDR20	4	29	30	4	HADDR4
HADDR19	3	31	32	3	HADDR3
HADDR18	2	33	34	2	HADDR2
HADDR17	1	35	36	1	HADDR1
HADDR16	0	37	38	0	HADDR0

Chapter 3: Preparing the Target System for AMBA AHB
Signal-to-Connector Mappings - AMBA AHB

STAT Signals - Single Master Configuration

LA Connector C2 Even LA Pod			LA Connector C2 Odd LA Pod		
ARM AMBA AHB Signal Name	LA Bit	MICTOR Connector Pin	MICTOR Connector Pin	LA Bit	ARM AMBA AHB Signal Name
no connect		1	2		no connect
no connect		3	4		no connect
unused	M clk	5	6	L clk	unused
unused	15	7	8	15	unused
unused	14	9	10	14	unused
HBUSREQ	13	11	12	13	unused
HGRANT	12	13	14	12	unused
HLOCK	11	15	16	11	unused
HPROT3	10	17	18	10	unused
HPROT2	9	19	20	9	unused
HRESP1	8	21	22	8	unused
HRESP0	7	23	24	7	unused
HREADY	6	25	26	6	unused
unused	5	27	28	5	unused
HSIZE1	4	29	30	4	HBURST2
HSIZE0	3	31	32	3	HBURST1
HPROT1	2	33	34	2	HBURST0
HPROTO	1	35	36	1	HTRANS1
HWRITE	0	37	38	0	HTRANS0

HRDATA Signals - Single Master Configuration

LA Connector C3 Even LA Pod			LA Connector C3 Odd LA Pod		
ARM AMBA AHB Signal Name	LA Bit	MICTOR Connector Pin	MICTOR Connector Pin	LA Bit	ARM AMBA AHB Signal Name
no connect		1	2		no connect
no connect		3	4		no connect
HREADY	K clk	5	6	J clk	unused
HRDATA31	15	7	8	15	HRDATA15
HRDATA30	14	9	10	14	HRDATA14
HRDATA29	13	11	12	13	HRDATA13
HRDATA28	12	13	14	12	HRDATA12
HRDATA27	11	15	16	11	HRDATA11
HRDATA26	10	17	18	10	HRDATA10
HRDATA25	9	19	20	9	HRDATA9
HRDATA24	8	21	22	8	HRDATA8
HRDATA23	7	23	24	7	HRDATA7
HRDATA22	6	25	26	6	HRDATA6
HRDATA21	5	27	28	5	HRDATA5
HRDATA20	4	29	30	4	HRDATA4
HRDATA19	3	31	32	3	HRDATA3
HRDATA18	2	33	34	2	HRDATA2
HRDATA17	1	35	36	1	HRDATA1
HRDATA16	0	37	38	0	HRDATA0

Chapter 3: Preparing the Target System for AMBA AHB
Signal-to-Connector Mappings - AMBA AHB

HWDATA Signals - Single Master Configuration

LA Connector C4 Even LA Pod			LA Connector C4 Odd LA Pod		
ARM AMBA AHB Signal Name	LA Bit	MICTOR Connector Pin	MICTOR Connector Pin	LA Bit	ARM AMBA AHB Signal Name
no connect		1	2		no connect
no connect		3	4		no connect
unused	M clk	5	6	L clk	unused
HWDATA31	15	7	8	15	HWDATA15
HWDATA30	14	9	10	14	HWDATA14
HWDATA29	13	11	12	13	HWDATA13
HWDATA28	12	13	14	12	HWDATA12
HWDATA27	11	15	16	11	HWDATA11
HWDATA26	10	17	18	10	HWDATA10
HWDATA25	9	19	20	9	HWDATA9
HWDATA24	8	21	22	8	HWDATA8
HWDATA23	7	23	24	7	HWDATA7
HWDATA22	6	25	26	6	HWDATA6
HWDATA21	5	27	28	5	HWDATA5
HWDATA20	4	29	30	4	HWDATA4
HWDATA19	3	31	32	3	HWDATA3
HWDATA18	2	33	34	2	HWDATA2
HWDATA17	1	35	36	1	HWDATA1
HWDATA16	0	37	38	0	HWDATA0

Signal-to-Connector Mappings — Multiple Master Configuration

The tables below list the signal mappings to high-density connectors for an ARM AMBA AHB system in the multiple master configuration. The logic analyzer bit used is listed as a reference. Use these tables to route ARM signals to the connector.

NOTE:

HREADY and HTRANS1 appear twice in the connector pinout definition. The inverse assembler uses the STAT connector pin 25 for HREADY and the STAT connector pin 36 for HTRANS1.

Chapter 3: Preparing the Target System for AMBA AHB
Signal-to-Connector Mappings - AMBA AHB

ADDR Signals - Multiple Master Configuration

LA Connector C1 Even LA Pod			LA Connector C1 Odd LA Pod		
ARM AMBA AHB Signal Name	LA Bit	MICTOR Connector Pin	MICTOR Connector Pin	LA Bit	ARM AMBA AHB Signal Name
no connect		1	2		no connect
no connect		3	4		no connect
HTRANS1	K clk	5	6	J clk	HCLK
ADDR31	15	7	8	15	ADDR15
ADDR30	14	9	10	14	ADDR14
ADDR29	13	11	12	13	ADDR13
ADDR28	12	13	14	12	ADDR12
ADDR27	11	15	16	11	ADDR11
ADDR26	10	17	18	10	ADDR10
ADDR25	9	19	20	9	ADDR9
ADDR24	8	21	22	8	ADDR8
ADDR23	7	23	24	7	ADDR7
ADDR22	6	25	26	6	ADDR6
ADDR21	5	27	28	5	ADDR5
ADDR20	4	29	30	4	ADDR4
ADDR19	3	31	32	3	ADDR3
ADDR18	2	33	34	2	ADDR2
ADDR17	1	35	36	1	ADDR1
ADDR16	0	37	38	0	ADDR0

STAT Signals - Multiple Master Configuration

LA Connector C2 Even LA Pod			LA Connector C2 Odd LA Pod		
ARM AMBA AHB Signal Name	LA Bit	MICTOR Connector Pin	MICTOR Connector Pin	LA Bit	ARM AMBA AHB Signal Name
no connect		1	2		no connect
no connect		3	4		no connect
unused	M clk	5	6	L clk	unused
unused	15	7	8	15	unused
unused	14	9	10	14	unused
unused	13	11	12	13	unused
unused	12	13	14	12	unused
unused	11	15	16	11	unused
HPROT3	10	17	18	10	HSPLIT5
HPROT2	9	19	20	9	HSPLIT4
HRESP1	8	21	22	8	HSPLIT3
HRESP0	7	23	24	7	HSPLIT2
HREADY	6	25	26	6	HSPLIT1
HMASTLOCK	5	27	28	5	HSPLIT0
HSIZE1	4	29	30	4	HBURST2
HSIZE0	3	31	32	3	HBURST1
HPROT1	2	33	34	2	HBURST0
HPROT0	1	35	36	1	HTRANS1
HWRITE	0	37	38	0	HTRANS0

Chapter 3: Preparing the Target System for AMBA AHB
Signal-to-Connector Mappings - AMBA AHB

HRDATA Signals - Multiple Master Configuration

LA Connector C3 Even LA Pod			LA Connector C3 Odd LA Pod		
ARM AMBA AHB Signal Name	LA Bit	MICTOR Connector Pin	MICTOR Connector Pin	LA Bit	ARM AMBA AHB Signal Name
no connect		1	2		no connect
no connect		3	4		no connect
HREADY	K clk	5	6	J clk	unused
DATA31	15	7	8	15	DATA15
DATA30	14	9	10	14	DATA14
DATA29	13	11	12	13	DATA13
DATA28	12	13	14	12	DATA12
DATA27	11	15	16	11	DATA11
DATA26	10	17	18	10	DATA10
DATA25	9	19	20	9	DATA9
DATA24	8	21	22	8	DATA8
DATA23	7	23	24	7	DATA7
DATA22	6	25	26	6	DATA6
DATA21	5	27	28	5	DATA5
DATA20	4	29	30	4	DATA4
DATA19	3	31	32	3	DATA3
DATA18	2	33	34	2	DATA2
DATA17	1	35	36	1	DATA1
DATA16	0	37	38	0	DATA0

HWDATA Signals - Multiple Master Configuration

LA Connector C4 Even LA Pod			LA Connector C4 Odd LA Pod		
ARM AMBA AHB Signal Name	LA Bit	MICTOR Connector Pin	MICTOR Connector Pin	LA Bit	ARM AMBA AHB Signal Name
no connect		1	2		no connect
no connect		3	4		no connect
unused	M clk	5	6	L clk	unused
unused	15	7	8	15	unused
unused	14	9	10	14	unused
unused	13	11	12	13	unused
unused	12	13	14	12	unused
unused	11	15	16	11	unused
unused	10	17	18	10	unused
unused	9	19	20	9	unused
unused	8	21	22	8	unused
unused	7	23	24	7	HLOCK3
MASTER2	6	25	26	6	HLOCK2
MASTER1	5	27	28	5	HLOCK1
MASTER0	4	29	30	4	HLOCK0
HGRANT3	3	31	32	3	HBUSREQ3
HGRANT2	2	33	34	2	HBUSREQ2
HGRANT1	1	35	36	1	HBUSREQ1
HGRANT0	0	37	38	0	HBUSREQ0

Designing a JTAG Connector into Your Target System

For information on designing a JTAG connector into your target system, see the emulation manual supplied with your emulation probe/module.

Setting Up the Logic Analysis System

Power-on/Power-off Sequence

Listed below are the sequences for powering on and off a fully connected system. Simply stated, your target system is always the last to be powered on, and the first to be powered off.

To power on Agilent 16700-series logic analysis systems

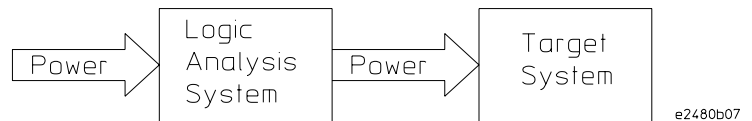
Ensure the target system is powered off.

- 1** Turn on the logic analyzer. The Setup Assistant will guide you through the process of configuring the logic analyzer and making connections from the logic analyzer to the target system.
- 2** When the logic analyzer is connected to the target system and everything is configured, turn on your target system.

To power on all other logic analyzers

With all components connected, power on your system in the following order:

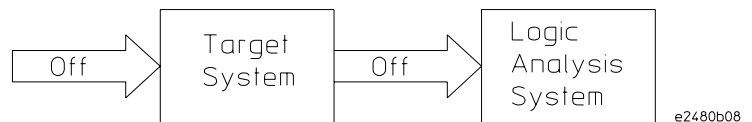
- 1** Logic analysis system.
- 2** Your target system.



To power off

Turn off power to your system in the following order:

- 1** Turn off your target system.
- 2** Turn off your logic analysis system.



Installing Logic Analyzer Modules

You should install logic analyzer, oscilloscope, or pattern generator modules in your logic analysis system before you install an emulation module (if applicable) and software.

CAUTION:

Electrostatic discharge (ESD) can damage electronic components. Use appropriate ESD equipment (grounded wrist strap, etc.) and ESD-safe procedures when you handle and install modules.

Refer to your logic analysis system's *Installation Guide* for instructions on installing logic analyzer modules.

Installing the Emulation Module

If you ordered an emulation module as part of your Agilent 16700-series logic analysis system, it is already installed in the system frame.

If you ordered your emulation module separately, then follow the instructions provided in the *Emulation for the ARM7/ARM9* user's guide to install your emulation module.

Installing Software

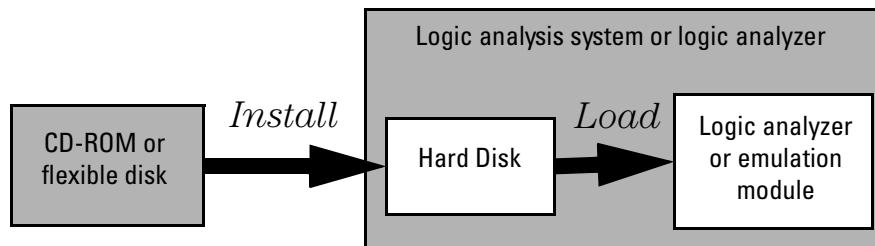
This section explains how to install the processor-specific software you will need for your logic analyzer.

NOTE:

If you ordered an emulation solution with your logic analysis system, the software was installed at the factory.

Installing and loading

Installing the software will copy the files from CD-ROM to the hard disk of your logic analysis system. Later, you will need to **load** some of the files into the appropriate measurement module.



What needs to be installed

Agilent 16700-series logic analysis systems

The following files are installed when you install a processor support package from the CD-ROM:

- Logic analysis system configuration files
- Inverse assembler (automatically loaded with the configuration files)
- Personality files for the Setup Assistant
- Emulation module firmware (for emulation solutions)
- Emulation Control Interface (for emulation solutions)

The Agilent B4620B Source Correlation Tool Set is installed with the logic analysis system's operating system.

To install the software from CD-ROM


Installing a processor support package from a CD-ROM will take just a few minutes. If the processor support package requires an update to the Agilent Technologies 16700 operating system, installation may take approximately 15 minutes.

If the CD-ROM drive is not connected, see the instructions printed on the CD-ROM package.

- 1 Turn on the CD-ROM drive first and then turn on the logic analysis system.

If the CD-ROM and analysis system are already turned on, be sure to save any acquired data. The installation process may reboot the logic analysis system.

- 2 Insert the CD-ROM in the drive.

- 3 Select the **System Administration** icon. 

- 4 Select the **Software Install** tab.

- 5 Select **Install...**

Change the media type to “CD-ROM” if necessary.

- 6 Select **Apply**.

- 7 From the list of types of packages, double-click “PROC-SUPPORT.”

NOTE:

For touch screen systems, double select the “PROC-SUPPORT” line by quickly touching it twice.

A list of the processor support packages on the CD-ROM will be displayed.

- 8 Select on the “ARM” package.

If you are unsure whether this is the correct package, select **Details** for information about the contents of the package.

- 9 Select **Install**.

The Continue dialog box will appear.

- 10 Select **Continue**.

Installing Software

The Software Install dialog will display “Progress: completed successfully” when the installation is complete.

- 11** If required, the system will automatically reboot. Otherwise, close the software installation windows.

The configuration files are stored in `/logic/configs/hp/arm`. The inverse assemblers are stored in `/logic/ia`.

See Also

The instructions printed on the CD-ROM package for a summary of the installation instructions.

The online help for more information on installing, licensing, and removing software.

Probing the Target System

Connecting the Logic Analyzer to the Target System

When you have chosen a connector type, and target system signals have been routed to connector headers according to the tables in chapter 2 or chapter 3, the target system can be connected to the logic analyzer.

NOTE:

High density connectors must be used for AMBA AHB inverse assembly. Medium density *or* high density connectors may be used for ARM core or AMBA ASB inverse assembly.

Each table on the following pages corresponds to a particular logic analyzer and contains entries for medium density and/or high density connectors. Also listed is the configuration file that is loaded into the analyzer for a correct mapping of target signals.

- Locate your logic analyzer on the following pages and connect your target system to the logic analyzer pod connectors.
 - Medium density connectors require Agilent Technologies 01650-63203 termination adapters.
 - High density connectors require Agilent Technologies E5346A adapter cables.

CAUTION:

Be sure to power down the target system before connecting or disconnecting cables. Otherwise, you may damage circuitry in the analyzer or target system.

Connecting the Logic Analyzer to the Target System

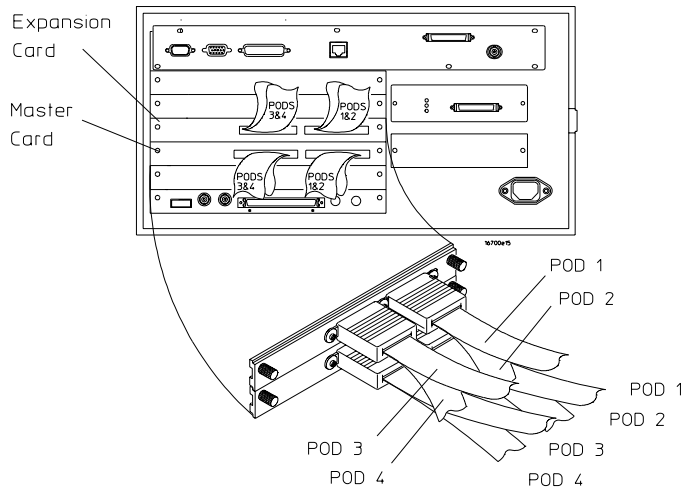
If you are connecting to an Agilent 16700-series logic analysis system, and have designed connectors into the target system as described in chapter 2 or chapter 3, use the Setup Assistant to connect and configure your system (see page 22). See also “Number of logic analyzer pods required” on page 30.

This section shows the connections between the connectors on your target system and the logic analyzer pod cables. Use the appropriate page for your logic analyzer.

Logic Analyzer Model Number	Number of cards installed (if applicable)	See connection diagram on page
16750/51/52A	2+ cards	124
16750/51/52A	1 card	125
16715/16/17/18/19A	2+ cards	124
16715/16/17/18/19A	1 card	125
16710/11/12A	2+ cards	126
16710/11/12A	1 card	127
16603A	n/a	128
16602A	n/a	129
16601A	n/a	130
16600A	n/a	131
16554/55/56/57	2 cards	132
16554/55/56/57	1 card	133
16550A	2+ cards	134
16550A	1 card	135
1671A/D/E	n/a	136
1670A/D/E	n/a	137
1661A/AS/C/CS/CP/E/ES/EP	n/a	138
1660A/AS/C/CS/CP/E/ES/EP	n/a	139

If you have an Agilent 16700-series logic analysis system with a logic analyzer card not listed here, use the Setup Assistant to connect and configure your logic analyzer.

To connect to a 16715/16/17/18/19A or 16750/51/52A logic analyzer (two cards)

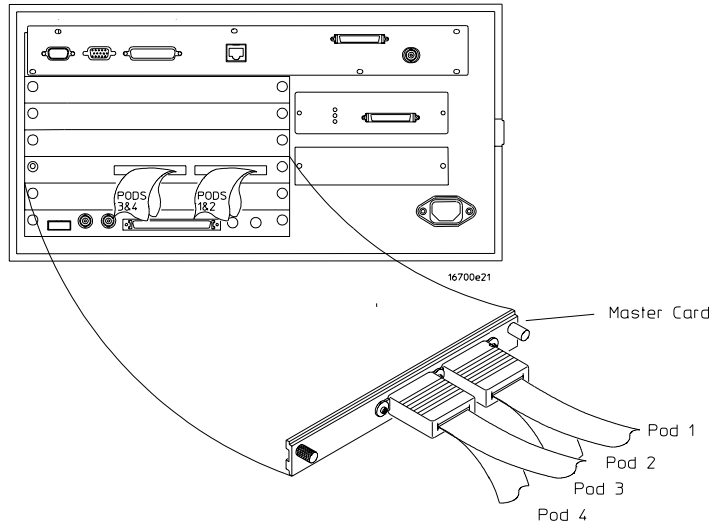


Use this table to connect cables from your target system headers to the analyzer. The tables are oriented similar to the analyzer's back panel.

Expansion Card	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	User Defined	User Defined	C6	C5
High Density Connector	C4-Even	C4-Odd	C3-Even	C3-Odd
Master Card	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	C4	C3	C2	C1
High Density Connector	C2-Even	C2-Odd	C1-Even	C1-Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core, page 63 for AMBA ASB and page 102 for AMBA AHB. You can install additional cards if you want to analyze additional signals.

To connect to a 16715/16/17/18/19A or 16750/51/52A logic analyzer (one card)



Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

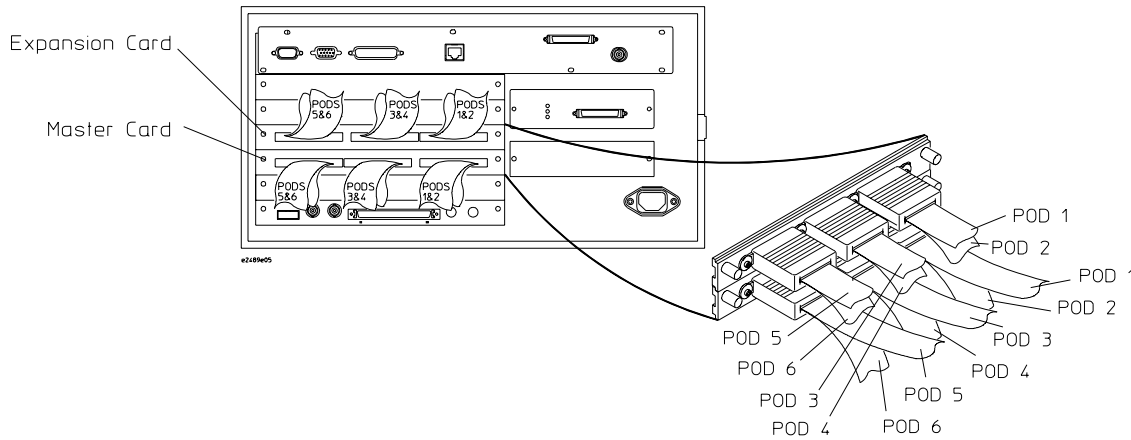
Logic Analyzer Card	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	C4	C3	C2	C1
High Density Connector	C2-Even	C2-Odd	C1-Even	C1-Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

NOTE:

ARM core or AMBA ASB inverse assembly can be performed with four pods if the target processor is using an 8-bit data bus or reduced address mode. In reduced address mode 24 address bits and 16 data bits are available for analysis. Reduced address mode uses a pinout which is different from other configurations.

To connect to a 16710/11/12A logic analyzer (two card)



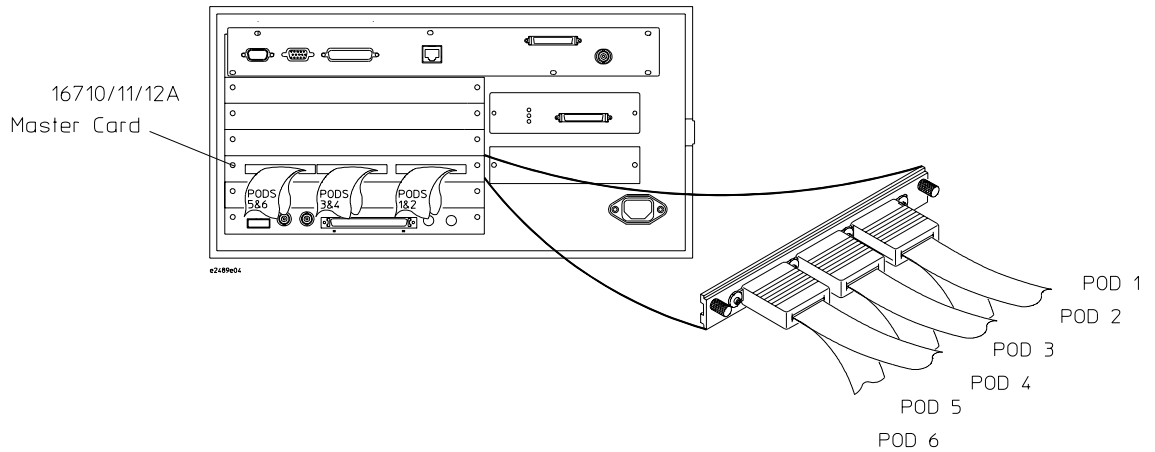
Use this table to connect cables from your target system headers to the analyzer. The tables are oriented similar to the analyzer's back panel.

Expansion Card	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
High Density Connector	User Defined	User Defined	User Defined	User Defined	C4-Even	C4-Odd

Master Card	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
High Density Connector	C1-Odd	C1-Even	C2-Odd	C2-Even	C3-Odd	C3-Even

The connector numbers (C1 Even, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 102 for AMBA AHB. You can install additional cards if you want to analyze additional signals.

To connect to a 16710/11/12A logic analyzer (one card)

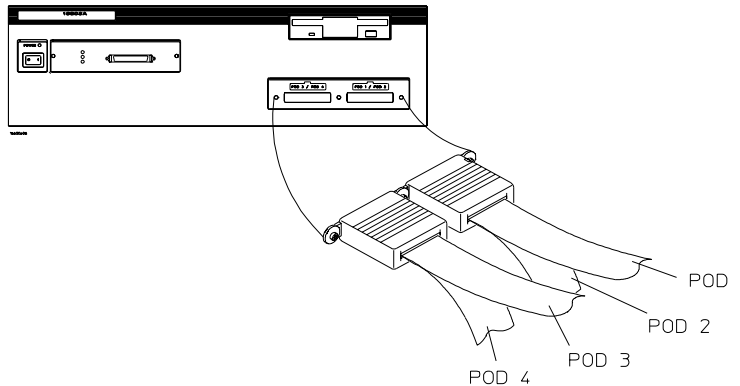


Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

Logic Analyzer Card	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	C6	C5	C4	C3	C2	C1
High Density Connector	C6-Even	C5-Odd	C4-Even	C3-Odd	C2-Even	C1-Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

To connect to a 16603A logic analyzer



Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

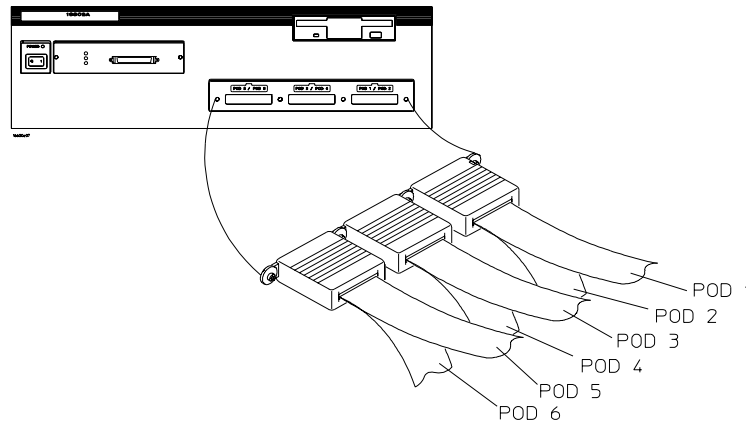
16603A Pod	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	C4	C3	C2	C1
High Density Connector	C2- Even	C2- Odd	C1- Even	C1- Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

NOTE:

ARM core or AMBA ASB inverse assembly can be performed with four pods if the target processor is using an 8-bit data bus or reduced address mode. In reduced address mode 24 address bits and 16 data bits are available for analysis. Reduced address mode uses a pinout which is different from other configurations.

To connect to a 16602A logic analyzer

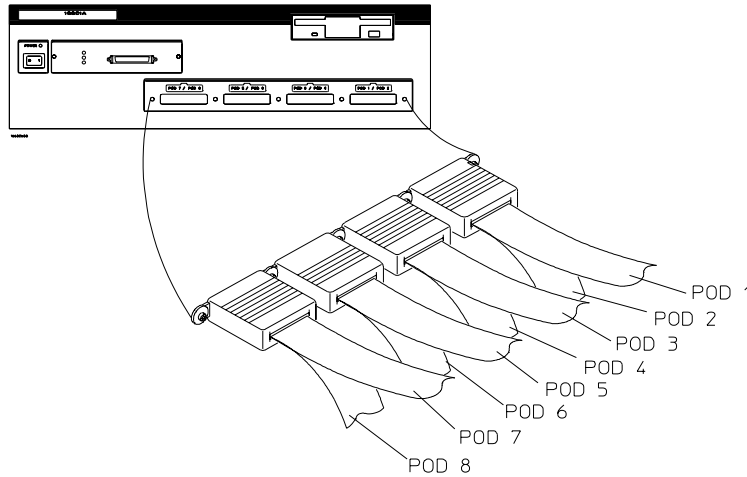


Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

16602A Pod	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	C6	C5	C4	C3	C2	C1
High Density Connector	C3-Even	C3-Odd	C2-Even	C2-Odd	C1-Even	C1-Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

To connect to a 16601A logic analyzer

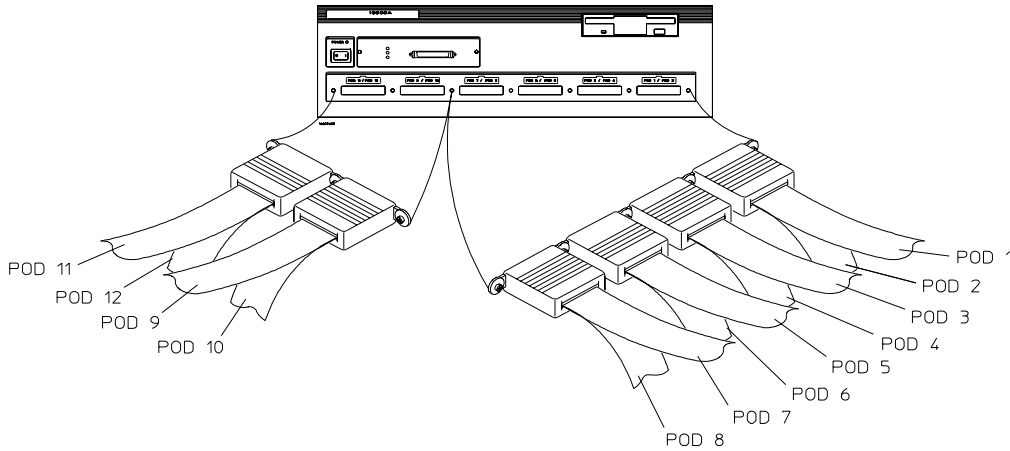


Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

16601A Pod	Pod 8	Pod 7	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	Not Used	Not Used	C6	C5	C4	C3	C2	C1
High Density Connector	C4-Even	C4-Odd	C3-Even	C3-Odd	C2-Even	C2-Odd	C1-Even	C1-Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

To connect to a 16600A logic analyzer

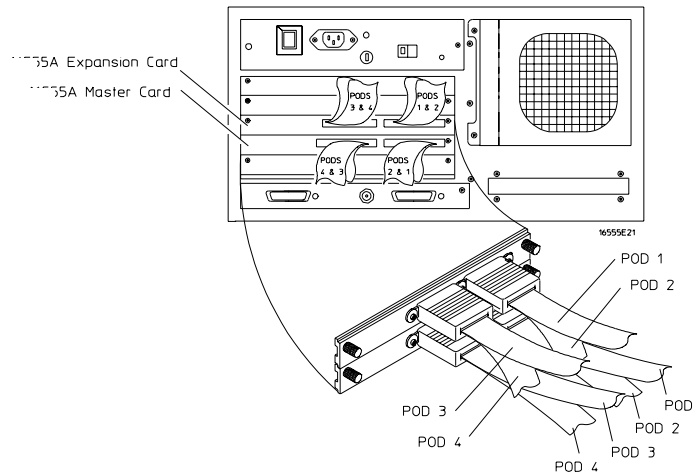


Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

16600A Pod	Pods 9-12	Pod 8	Pod 7	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	Not Used	Not Used	Not Used	C6	C5	C4	C3	C2	C1
High Density Connector	Not Used	C4-Even	C4-Odd	C3-Even	C3-Odd	C2-Even	C2-Odd	C1-Even	C1-Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

To connect to a 16554/55/56/57 logic analyzer (two-card)



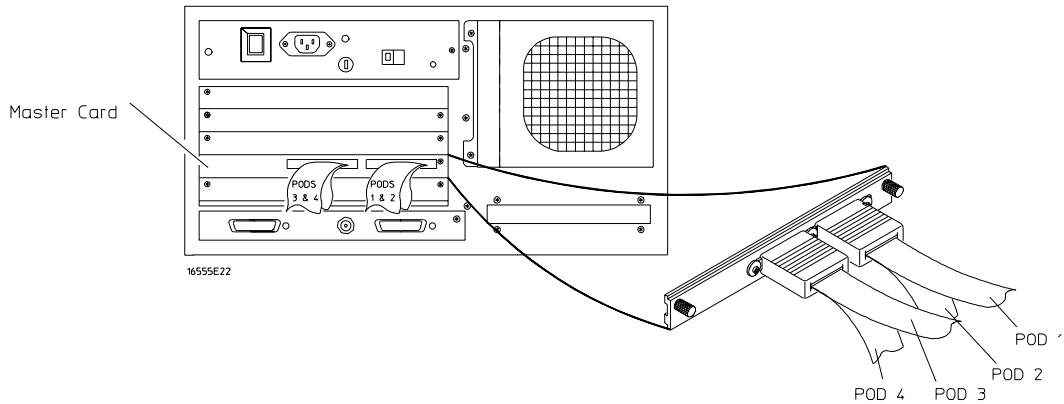
Use these tables to connect cables from your target system headers to the analyzer. The tables are oriented similar to the analyzer's back panel.

Expansion Card	Pod 4	Pod 3	Pod 2	Pod1
Medium Density Connector	Not Used	Not Used	C6	C5
High Density Connector	C4-Even	C4-Odd	C3-Even	C3-Odd

Master Card	Pod 4	Pod 3	Pod 2	Pod1
Medium Density Connector	C4	C3	C2	C1
High Density Connector	C2-Even	C2-Odd	C1-Even	C1-Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core, page 63 for AMBA ASB and page 102 for AMBA AHB. You can install additional cards if you want to analyze additional signals.

To connect to a 16554/55/56/57 analyzer (one-card)



Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

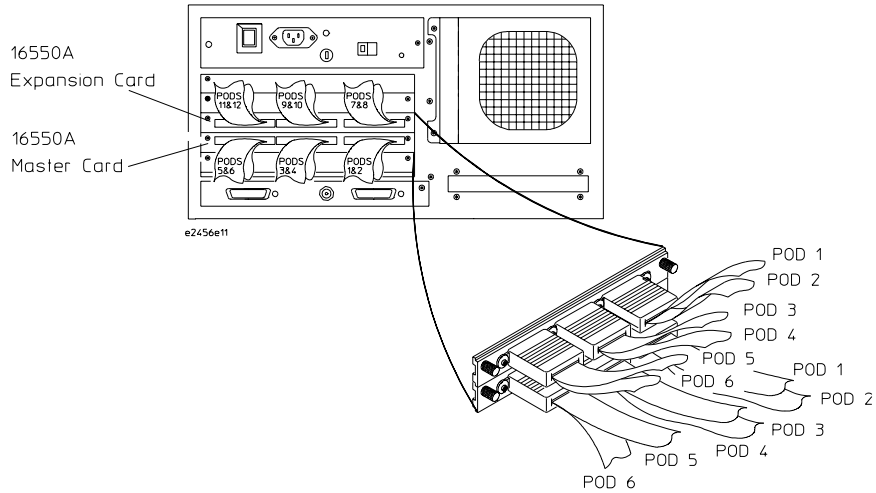
16554/55/56/57 Master Card Pod	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	C4	C3	C2	C1
High Density Connector	C2- Even	C2- Odd	C1- Even	C1- Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

NOTE:

ARM core or AMBA ASB inverse assembly can be performed with four pods if the target processor is using an 8-bit data bus or reduced address mode. In reduced address mode 24 address bits and 16 data bits are available for analysis. Reduced address mode uses a pinout which is different from other configurations.

To connect to a 16550A logic analyzer (two card)



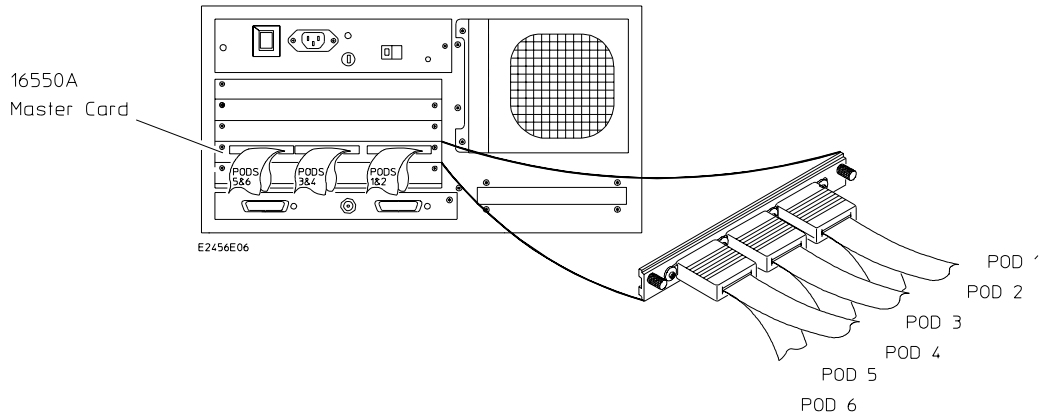
Use these tables to connect cables from your target system headers to the analyzer. The tables are oriented similar to the analyzer's back panel.

16550A Pod	Pod 6	Pod 5	Pod 9	Pod 3	Pod 2	Pod 1
Medium Density Connector	Not Used	Not Used	Not Used	Not Used	C8	C7
High Density Connector	Not Used	Not Used	Not Used	Not Used	C4-Even	C4-Odd

16550A Pod	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	C6	C5	C4	C3	C2	C1
High Density Connector	C3-Even	C3-Odd	C2-Even	C2-Odd	C1-Even	C1-Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core, page 63 for AMBA ASB and page 102 for AMBA AHB. You can install additional cards if you want to analyze additional signals.

To connect to a 16550A logic analyzer (one card)

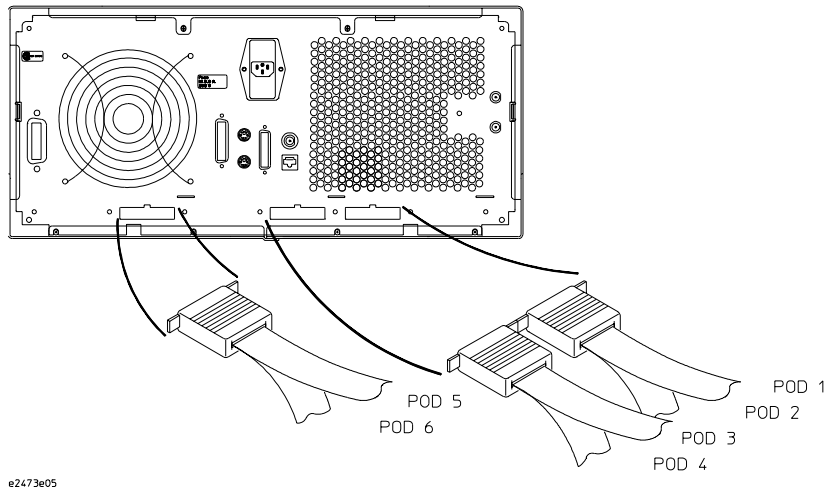


Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

16550A Pod	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	C6	C5	C4	C3	C2	C1
High Density Connector	C3-Even	C3-Odd	C2-Even	C2-Odd	C1-Even	C1-Odd

The connector numbers (C1, C2 Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

To connect to a 1671A/D/E logic analyzer



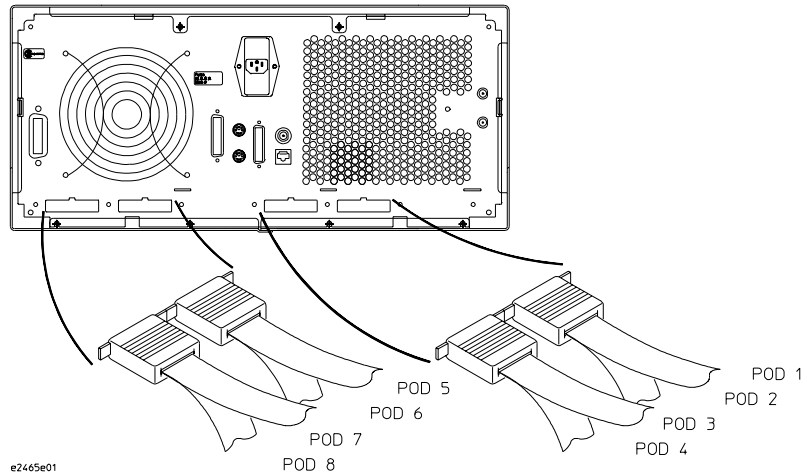
e2473e05

Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

1671 Pod	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	C6	C5	C4	C3	C2	C1
High Density Connector	C3- Even	C3- Odd	C2- Even	C2- Odd	C1- Even	C1- Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

To connect to a 1670A/D/E logic analyzer

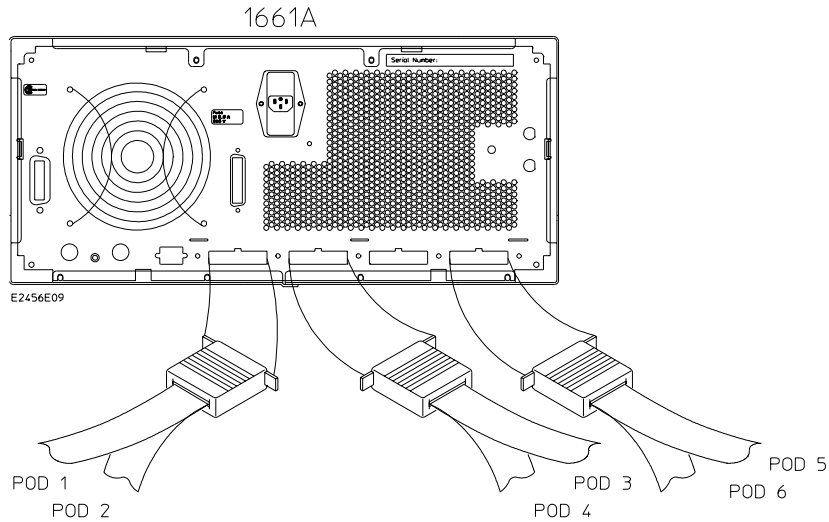


Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

1670 Pod	Pod 8	Pod 7	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1
Medium Density Connector	User Defined	User Defined	C6	C5	C4	C3	C2	C1
High Density Connector	User Defined	User Defined	C3-Even	C3-Odd	C2-Even	C2-Odd	C1-Even	C1-Odd

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

To connect to a 1661A/AS/C/CS/E/ES/EP logic analyzer

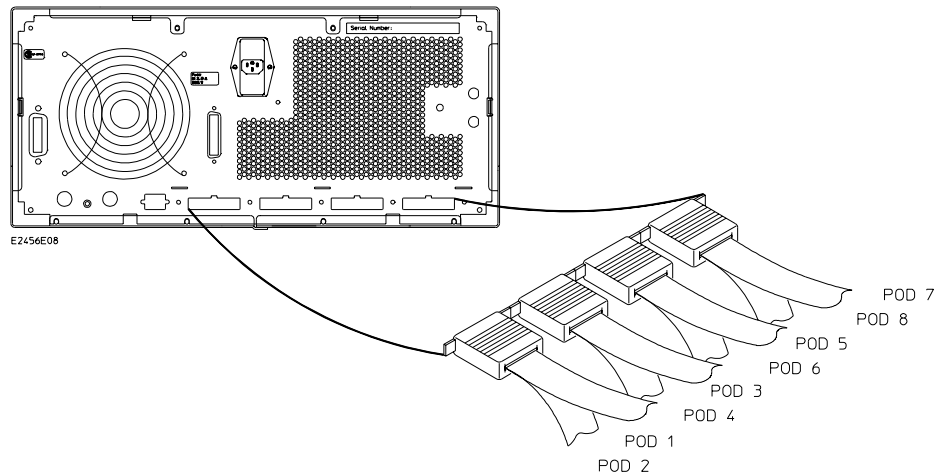


Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

1661 Pod	Pod 1	Pod 2	Pod 3	Pod 4	Pod 5	Pod 6
Medium Density Connector	C1	C2	C3	C4	C5	C6
High Density Connector	C1- Odd	C1- Even	C2- Odd	C2- Even	C3- Odd	C3- Even

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

To connect to a 1660A/AS/C/CS/E/ES/EP logic analyzer



Use this table to connect cables from your target system headers to the analyzer. The table is oriented similar to the analyzer's back panel.

1660 Pod	Pod 1	Pod 2	Pod 3	Pod 4	Pod 5	Pod 6	Pod 7	Pod 8
Medium Density Connector	C1	C2	C3	C4	C5	C6	User Defined	User Defined
High Density Connector	C1-Odd	C1-Even	C2-Odd	C2-Even	C3-Odd	C3-Even	User Defined	User Defined

The connector numbers (C1, C2-Odd, etc.) correspond to the signal-to-connector mappings starting on page 49 for ARM core or page 63 for AMBA ASB. You can install additional cards if you want to analyze additional signals.

Chapter 5: Probing the Target System
Connecting the Logic Analyzer to the Target System

Configuring the 16700-series Logic
Analysis System

Chapter 6: Configuring the 16700-series Logic Analysis System

Configuring 16700-series Logic Analysis Systems

The sections of this chapter describe setting up and using the ARM inverse assembler. Because your ARM target system is designed uniquely according to your needs, it is important that you specify the available signals and memory regions to the inverse assembler.

The information in this chapter is presented in the following sections:

- Loading the configuration file and the inverse assembler
- Tables showing configuration file names
- Inverse assembler modes of operation
- Using the Invasm menu
- Setting the inverse assembler preferences
- Symbols
- Compilers

Configuring 16700-series Logic Analysis Systems

You configure the logic analyzer by loading a configuration file. Normally this is done using the Setup Assistant (see page 22). If you did not use the Setup Assistant, you can load the configuration and inverse assembler files from the logic analysis system hard disk.

The information in the configuration file includes:

- Label names and channel assignments for the logic analyzer
- Inverse assembler file name

The configuration file you use is determined by the logic analyzer you are using, and whether you are performing state or timing analysis.

The procedures for loading a configuration file depend on the type of logic analyzer you are using. This chapter describes configuration of Agilent 16700-series logic analysis systems. See Chapter 7, “Configuring the 1660A/1670A/16500B/C-Series Logic Analyzer,” beginning on page 169, for information about setting up 1660/1670/16500-series logic analyzers.

To load configuration files (and the inverse assembler) from hard disk—16700-series logic analysis systems

If you use Setup Assistant, it will load configuration files and the inverse assembler for you. This is the preferred method. If you did not use Setup Assistant, you can load the configuration and inverse assembler files from the logic analysis system hard disk.

- 1** Select the File Manager icon. Use File Manager to ensure that the subdirectory `/hplogic/configs/hp/arm/` exists.

If the above directory does not exist, you need to install the ARM Processor Support Package. Close File Manager, then use the procedure on the CD-ROM jacket to install the ARM Processor Support Package before you continue. See “Installing Software” on page 118 for details.

- 2** Using File Manager, select the configuration file you want to load in the `/hplogic/configs/hp/arm/` directory, then select **Load**. If you have more than one logic analyzer installed in your logic analysis system, use the Target field to select the machine you want to load.

The logic analyzer is configured for ARM analysis by loading the appropriate ARM configuration file. Loading the indicated file also automatically loads the inverse assembler. The configuration file names are shown in the following table.

- 3** Close File Manager.

To load configuration files (and the inverse assembler) from floppy disk—16700-series logic analysis systems

If you use Setup Assistant, it will load configuration files and the inverse assembler for you. This is the preferred method. If you did not use Setup Assistant, you can load the configuration and inverse assembler files from the logic analysis system hard disk or floppy disk; however, the preferred method is to install this functionality from the CD-ROM onto the hard disk and load from the hard disk.

To install a configuration and inverse assembler file from a floppy disk:

- 1** Insert the floppy disk in the floppy drive on the Agilent 16700-series logic analysis system mainframe.
- 2** In the logic analysis System window, select the File Manager icon.
- 3** In the File Manager window:
 - Set Current Disk to **Flexible Disk**.
 - Set Target to the analyzer you wish to configure.
 - Select the name of the desired configuration file in the Contents frame. The Contents frame lists the configuration files and inverse assembler files available on the floppy disk. These may be either DOS or LIF format files. Either format can be loaded directly into the appropriate logic analyzers.

Note that the logic analyzers read both DOS and LIF formats. However, only DOS formatted floppy disks can be used to store configurations and data. LIF format floppy disks are read-only.

- 4** Select **Load**.

The configuration file you choose will set up the logic analyzer and associated tools. You may see Information, Error, and Warning dialogs that say your configuration has been loaded, and advise you about making proper connections.
- 5** Select the Workspace window icon to see the arrangement of analysis tools in your configuration.
- 6** Right-click the logic analyzer icon in your configuration and choose its **Setup** button to see the way your configuration file defined the Config, Format, and Trigger options.

NOTE:

Under the **Format** tab, buses are labeled, and bits included in each bus are identified by an asterisk "*".

This procedure restores the configuration that was in effect when the configuration file was saved. Because the file was not saved using your system, you may receive error messages about loading the enhanced inverse assembler or about pods that were truncated. Select the Config, Format, and Trigger tabs and modify the configuration to satisfy your measurement desires. Then you can save your customized configuration to DOS format using the File→Save Configuration selection in any of your tool windows, or selecting the Save tab in the File Manager. For details about how to save configuration files, open the Help window.

To list software packages that are installed (16700-series logic analysis system)

- In the System Administration Tools window, select List....

ARM7 core analysis configuration files

Analyzer Model Number	Analyzer Module Description	Memory Controller			
		ARM 8-bit Normal Analysis	ARM 8-bit Reverse Analysis	ARM 16/32-bit Normal Analysis	ARM 16/32-bit Reverse Analysis
16750/51/52 (1 card)	400 MHz STATE 2 GHz TIMING ZOOM	CARM7L_1	CARM7ML_1	CARM7L_5*	CARM7ML_5*
16717/18/19 (1 card)	333 MHz STATE 2 GHz TIMING ZOOM	CARM7L_1	CARM7ML_1	CARM7L_5*	CARM7ML_5*
16716 (1 card)	167 MHz STATE 2 GHz TIMING ZOOM	CARM7L_1	CARM7ML_1	CARM7L_5*	CARM7ML_5*
16715 (1 card)	167 MHz STATE 667 MHz TIMING ZOOM	CARM7L_1	CARM7ML_1	CARM7L_5*	CARM7ML_5*
16750/51/52 (2 or more cards)	400 MHz STATE 2 GHz TIMING ZOOM	CARM7L_1	CARM7ML_1	CARM7L_2	CARM7ML_2
16717/18/19 (2 or more cards)	333 MHz STATE 2 GHz TIMING ZOOM	CARM7L_1	CARM7ML_1	CARM7L_2	CARM7ML_2
16716 (2 or more cards)	167 MHz STATE 2 GHz TIMING ZOOM	CARM7L_1	CARM7ML_1	CARM7L_2	CARM7ML_2
16715 (2 or more cards)	167 MHz STATE 667 MHz TIMING ZOOM	CARM7L_1	CARM7ML_1	CARM7L_2	CARM7ML_2
16710/11/12 (1 or more cards)	100 MHz STATE 250 MHz TIMING	CARM7_4	CARM7M_4	CARM7_3	CARM7M_3
16600A	n/a	CARM7_4	CARM7M_4	CARM7_3	CARM7M_3
16601A	n/a	CARM7_4	CARM7M_4	CARM7_3	CARM7M_3
16602A	n/a	CARM7_4	CARM7M_4	CARM7_3	CARM7M_3
16603A	n/a	CARM7_4	CARM7M_4	CARM7_5*	CARM7M_5*
16550A (1 or more cards)	100 MHz STATE 250 MHz TIMING	CARM7_4	CARM7M_4	CARM7_3	CARM7M_3
16554A (1 card)	0.5 M SAMPLE 70/250 MHz LA	CARM7_1	CARM7M_1	CARM7_5*	CARM7M_5*
16555A/D (1 card)	1.0 M SAMPLE 110/250 MHz LA	CARM7_1	CARM7M_1	CARM7_5*	CARM7M_5*
16556A/D (1 card)	1.0 M SAMPLE 100/400 MHz LA	CARM7_1	CARM7M_1	CARM7_5*	CARM7M_5*

Analyzer Model Number	Analyzer Module Description	Memory Controller			
		ARM 8-bit Normal Analysis	ARM 8-bit Reverse Analysis	ARM 16/32-bit Normal Analysis	ARM16/32-bit Reverse Analysis
16557D (1 card)	2.0 M SAMPLE 135/250 MHz LA	CARM7_1	CARM7M_1	CARM7_5*	CARM7M_5*
16554A (2 or more cards)	0.5 M SAMPLE 70/250 MHz LA	CARM7_1	CARM7M_1	CARM7_2	CARM7M_2
16555A/D (2 or more cards)	1.0 M SAMPLE 110/250 MHz LA	CARM7_1	CARM7M_1	CARM7_2	CARM7M_2
16556A/D (2 or more cards)	1.0 M SAMPLE 100/400 MHz LA	CARM7_1	CARM7M_1	CARM7_2	CARM7M_2
16557D (2 or more cards)	2.0 M SAMPLE 135/250 MHz LA	CARM7_1	CARM7M_1	CARM7_2	CARM7M_2
1660/70/71	n/a	CARM7_4	CARM7M_4	CARM7_3	CARM7M_3

*Reduced address mode

ARM 7 AMBA ASB analysis configuration files

Analyzer Model Number	Analyzer Module Description	Memory Controller			
		ARM 8-bit Normal Analysis	ARM 8-bit Reverse Analysis	ARM 16/32-bit Normal Analysis	ARM16/32-bit Reverse Analysis
16750/51/52 (1 card)	400 MHz STATE 2 GHz TIMING ZOOM	CAMBAL_1	CAMBAML_1	CAMBAL_5*	CAMBAML_5*
16717/18/19 (1 card)	333 MHz STATE 2 GHz TIMING ZOOM	CAMBAL_1	CAMBAML_1	CAMBAL_5*	CAMBAML_5*
16716 (1 card)	167 MHz STATE 2 GHz TIMING ZOOM	CAMBAL_1	CAMBAML_1	CAMBAL_5*	CAMBAML_5*
16715 (1 card)	167 MHz STATE 667 MHz TIMING ZOOM	CAMBAL_1	CAMBAML_1	CAMBAL_5*	CAMBAML_5*
16750/51/52 (2 or more cards)	400 MHz STATE 2 GHz TIMING ZOOM	CAMBAL_1	CAMBAML_1	CAMBAL_2	CAMBAML_2
16717/18/19 (2 or more cards)	333 MHz STATE 2 GHz TIMING ZOOM	CAMBAL_1	CAMBAML_1	CAMBAL_2	CAMBAML_2
16716 (2 or more cards)	167 MHz STATE 2 GHz TIMING ZOOM	CAMBAL_1	CAMBAML_1	CAMBAL_2	CAMBAML_2
16715 (2 or more cards)	167 MHz STATE 667 MHz TIMING ZOOM	CAMBAL_1	CAMBAML_1	CAMBAL_2	CAMBAML_2
16710/11/12 (1 or more cards)	100 MHz STATE 250 MHz TIMING	CAMBA_4	CAMBAM_4	CAMBA_3	CAMBAM_3
16600A	n/a	CAMBA_4	CAMBAM_4	CAMBA_3	CAMBAM_3
16601A	n/a	CAMBA_4	CAMBAM_4	CAMBA_3	CAMBAM_3
16602A	n/a	CAMBA_4	CAMBAM_4	CAMBA_3	CAMBAM_3
16603A	n/a	CAMBA_4	CAMBAM_4	CAMBA_5*	CAMBAM_5*
16550A (1 or more cards)	100 MHz STATE 250 MHz TIMING	CAMBA_4	CAMBAM_4	CAMBA_3	CAMBAM_3
16554A (1 card)	0.5 M SAMPLE 70/250 MHz LA	CAMBA_1	CAMBAM_1	CAMBA_5*	CAMBAM_5*
16555A/D (1 card)	1.0 M SAMPLE 110/250 MHz LA	CAMBA_1	CAMBAM_1	CAMBA_5*	CAMBAM_5*
16556A/D (1 card)	1.0 M SAMPLE 100/400 MHz LA	CAMBA_1	CAMBAM_1	CAMBA_5*	CAMBAM_5*

Analyzer Model Number	Analyzer Module Description	Memory Controller			
		ARM 8-bit Normal Analysis	ARM 8-bit Reverse Analysis	ARM 16/32-bit Normal Analysis	ARM16/32-bit Reverse Analysis
16557D (1 card)	2.0 M SAMPLE 135/250 MHz LA	CAMBA_1	CAMBAM_1	CAMBA_5*	CAMBAM_5*
16554A (2 or more cards)	0.5 M SAMPLE 70/250 MHz LA	CAMBA_1	CAMBAM_1	CAMBA_2	CAMBAM_2
16555A/D (2 or more cards)	1.0 M SAMPLE 110/250 MHz LA	CAMBA_1	CAMBAM_1	CAMBA_2	CAMBAM_2
16556A/D (2 or more cards)	1.0 M SAMPLE 100/400 MHz LA	CAMBA_1	CAMBAM_1	CAMBA_2	CAMBAM_2
16557D (2 or more cards)	2.0 M SAMPLE 135/250 MHz LA	CAMBA_1	CAMBAM_1	CAMBA_2	CAMBAM_2
1660/70/71	n/a	CAMBA_4	CAMBAM_4	CAMBA_3	CAMBAM_3

*Reduced address mode

ARM9 AMBA ASB analysis configuration files

Analyzer Model Number	Analyzer Module Description	Memory Controller	
		ARM 8-bit Analysis	ARM 16/32-bit Analysis
16750/51/52 (1 card)	400 MHz STATE 2 GHz TIMING ZOOM	CAMBA9L_1	CAMBA9L_5*
16717/18/19 (1 card)	333 MHz STATE 2 GHz TIMING ZOOM	CAMBA9L_1	CAMBA9L_5*
16716 (1 card)	167 MHz STATE 2 GHz TIMING ZOOM	CAMBA9L_1	CAMBA9L_5*
16715 (1 card)	167 MHz STATE 667 MHz TIMING ZOOM	CAMBA9L_1	CAMBA9L_5*
16750/51/52 (2 or more cards)	400 MHz STATE 2 GHz TIMING ZOOM	CAMBA9L_1	CAMBA9L_2
16717/18/19 (2 or more cards)	333 MHz STATE 2 GHz TIMING ZOOM	CAMBA9L_1	CAMBA9L_2
16716 (2 or more cards)	167 MHz STATE 2 GHz TIMING ZOOM	CAMBA9L_1	CAMBA9L_2
16715 (2 or more cards)	167 MHz STATE 667 MHz TIMING ZOOM	CAMBA9L_1	CAMBA9L_2
16710/11/12 (1 or more cards)	100 MHz STATE 250 MHz TIMING	CAMBA9_4	CAMBA9_3
16600A	n/a	CAMBA9_4	CAMBA9_3
16601A	n/a	CAMBA9_4	CAMBA9_3
16602A	n/a	CAMBA9_4	CAMBA9_3
16603A	n/a	CAMBA9_4	CAMBA9_5*
16550A (1 or more cards)	100 MHz STATE 250 MHz TIMING	CAMBA9_4	CAMBA9_3
16554A (1 card)	0.5 M SAMPLE 70/250 MHz LA	CAMBA9_1	CAMBA9_5*
16555A/D (1 card)	1.0 M SAMPLE 110/250 MHz LA	CAMBA9_1	CAMBA9_5*
16556A/D (1 card)	1.0 M SAMPLE 100/400 MHz LA	CAMBA9_1	CAMBA9_5*

Analyzer Model Number	Analyzer Module Description	Memory Controller	
		ARM 8-bit Analysis	ARM 16/32-bit Analysis
16557D (1 card)	2.0 M SAMPLE 135/250 MHz LA	CAMBA9_1	CAMBA9_5*
16554A (2 or more cards)	0.5 M SAMPLE 70/250 MHz LA	CAMBA9_1	CAMBA9_2
16555A/D (2 or more cards)	1.0 M SAMPLE 110/250 MHz LA	CAMBA9_1	CAMBA9_2
16556A/D (2 or more cards)	1.0 M SAMPLE 100/400 MHz LA	CAMBA9_1	CAMBA9_2
16557D (2 or more cards)	2.0 M SAMPLE 135/250 MHz LA	CAMBA9_1	CAMBA9_2
1660/70/71	n/a	CAMBA9_4	CAMBA9_3

*Reduced address mode

ARM AMBA AHB analysis configuration files

Analyzer Model Number	Analyzer Module Description	Memory Controller	
		Single Master Configuration	Multiple Master Configuration
16750/51/52 (2 or more cards)	400 MHz STATE 2 GHz TIMING ZOOM	CARMAHB_3	CARMAHB_4
16717/18/19 (2 or more cards)	333 MHz STATE 2 GHz TIMING ZOOM	CARMAHB_3	CARMAHB_4
16716 (2 or more cards)	167 MHz STATE 2 GHz TIMING ZOOM	CARMAHB_3	CARMAHB_4
16715 (2 or more cards)	167 MHz STATE 667 MHz TIMING ZOOM	CARMAHB_3	CARMAHB_4
16710/11/12 (2 or more cards)	100 MHz STATE 250 MHz TIMING	CARMAHB_5	CARMAHB_6
16557D (2 or more cards)	2.0 M SAMPLE 135/250 MHz LA	CARMAHB_1	CARMAHB_2
16556A/D (2 or more cards)	1.0 M SAMPLE 100/400 MHz LA	CARMAHB_1	CARMAHB_2
16555A/D (2 or more cards)	1.0 M SAMPLE 110/250 MHz LA	CARMAHB_1	CARMAHB_2
16554A (2 or more cards)	0.5 M SAMPLE 70/250 MHz LA	CARMAHB_1	CARMAHB_2
16550A (2 or more cards)	100 MHz STATE 250 MHz TIMING	CARMAHB_5	CARMAHB_6

Inverse Assembler Modes of Operation

The logic analyzer can be configured to capture target system data in either state or timing mode.

State mode

This is the default mode which is set up by the configuration files.

In state mode, the logic analyzer uses the MCLK (or BCLK) signal from the ARM target system to capture data synchronously. This mode allows inverse assembly of ARM instructions.

To change to timing mode

In Timing mode, the logic analyzer samples the incoming signals asynchronously. *Inverse assembly is not available in timing mode.* To configure your logic analyzer for timing analysis:

- 1 Select the logic analyzer icon.
 - 2 Select “**Setup...**” from the menu. The “**Sampling**” tab will be active on the window that appears.
 - 3 Select the **Timing Mode** button.
-

Disabling the cache

NOTE:

Instructions executed in ARM cache will not be subject to inverse assembly.

Certain versions of the ARM core may have internal cache. When the cache is enabled, many ARM instructions do not appear on the external bus. To get an execution trace on the bus (or device pins), the cache can be disabled. The accomplishment of this will vary based on your system’s design.

To use the Invasm menu

The Invasm menu provides four choices: Load, Preferences, Filter, and Options. Access the Invasm menu in the listing window.

You must use the Preferences dialog to configure the inverse assembler to match the microprocessor memory controller configuration. The Filter and Options dialogs assist in analyzing and displaying data.

Loading the Inverse Assembler

The Load dialog lets you load a different inverse assembler and apply it to the data in the Listing window. In some cases you may have acquired raw data; you can use the Load dialog to apply an inverse assembler to that data.

Unloading the Inverse Assembler

If desired, the inverse assembler can be unloaded by selecting **Invasm, Unload** in the Listing window. However, if you want to load a different inverse assembler, unloading the current inverse assembler is not necessary. Simply load a new configuration file or use the Setup Assistant to load the new configuration. The **Unload** option is primarily for diagnostic purposes.

Setting the Inverse Assembler Preferences

Why the configuration is necessary

The number and type of status signals available for probing is unique to each ARM custom application. To provide the best possible decoding, this information must be supplied to the inverse assembler software.

Because critical information about what type of data is being accessed through a memory bank (memory region) is stored in internal registers, the inverse assembler needs to be given information about how the memory system is set up.

The memory controller operates by mapping every address to one of eight memory banks. Each memory bank can be set up to drive different external signals, to have different write permissions, etc. The memory banks are numbered from 0 to 7. Memory bank 0 has the highest priority and bank 7 has the lowest.

Each memory bank has values that describe the width of the memory accessed through that bank, the type of data, and the addresses that will be accessed through that bank. Since this information is not given on external signals, the inverse assembler provides a preferences window to enter this information so that the data decode can be as accurate as possible.

After configuring the inverse assembler using the dialogs on the following pages, set up a trigger and run the logic analyzer to capture a data trace.

NOTE:

To capture meaningful data, the inverse assembler preferences must be set up before running the logic analyzer.

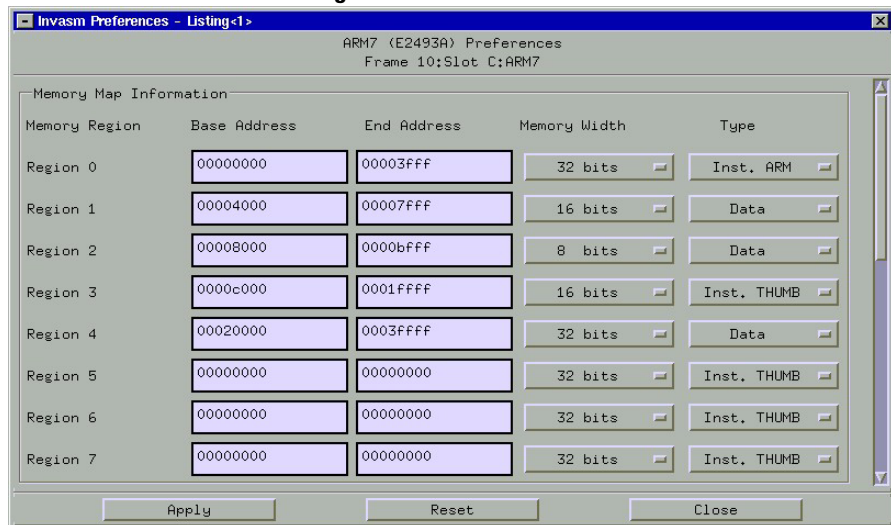
To set the memory map preferences — ARM core and AMBA ASB

The Memory Map dialog allows you to configure the inverse assembler to match the ARM memory configuration regarding bus widths and the type of data in each memory region.

A memory region is a logical block of memory with a homogeneous set of characteristics. A memory region does not equate to a physical memory device; a single physical memory device can contain several memory regions for code and data space.

An ARM system may not always provide signals to distinguish instruction reads from data reads or ARM instructions from Thumb instructions. Also, there are no standard signals for specifying the width of the memory bus. When decoding a given address, the ARM inverse assembler uses the mapping information in the Memory Map dialog to determine the access type and memory bus width.

AMBA ASB Preferences Dialog



Describe your target circuit memory regions as follows:

- For up to 8 memory regions in your target system, indicate the Base Address, End Address, data bus Width and Type of instruction.

- Types of memory accesses are: Inst. ARM, Inst. THUMB, and Data.
- If nOPC and MAS[1] are supplied by your system, the Type field is not required to determine the instruction type and is ignored.
- The inverse assembler assumes all memory regions are valid, so lower-numbered regions should be used before higher-numbered regions. Region 0 has the highest priority, and Region 7 has the lowest priority.
- If the inverse assembler returns “IA Error: Address not in map” then the address did not meet the specifications for any of the memory regions.

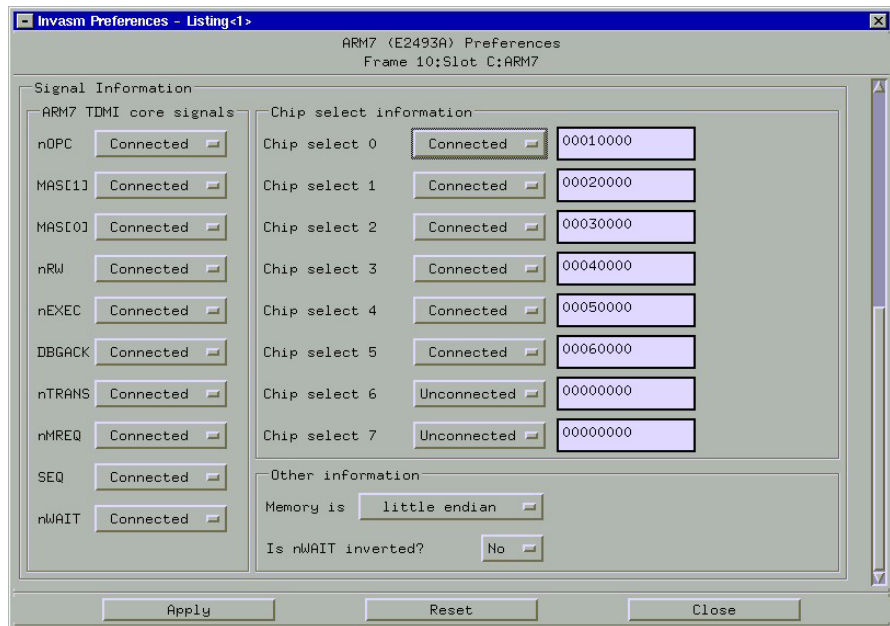
NOTE:

Regardless of the number of status bits available, the memory width must always be specified.

Signals Dialog — ARM core and AMBA ASB

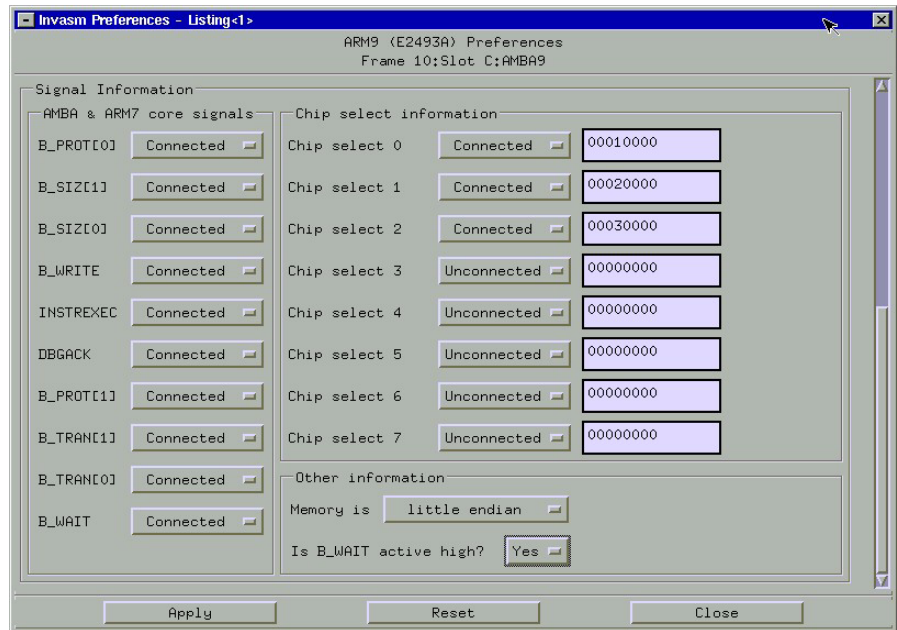
Use the Signals Dialog to tell the inverse assembler which target signals are available and connected to the logic analyzer. (To control which states are displayed see “Display Filtering” on page 218.) The signals listed in the dialog will differ based on whether you selected **ARM Core** or **AMBA Bus (ASB)** system when using the Setup Assistant. From the **Invasm** menu, select **Preferences** and scroll down.

Signals Dialog for an ARM core system



- Based on whether these signals are connected to the logic analyzer, select Connected or Unconnected for: nOPC, MAS[1], MAS[0], nRW, nEXEC, DBGACK, nTRANS, nMREQ, SEQ, nWAIT.
- Based on whether chip selects are used and connected to the logic analyzer, select Connected or Unconnected for each one, and indicate the base address to which the chip select refers.
- Indicate whether memory is little endian or big endian, or whether the BIGEND signal is being used.
- Indicate whether the nWAIT signal is inverted.

Signals Dialog for an AMBA ASB bus system



- Based on whether these signals are connected to the logic analyzer, select Connected or Unconnected for: B_PROT[0], B_SIZ[1], B_SIZ[0], B_WRITE, INSTREXEC, DBGACK, B_PROT[1], B_TRAN[1], B_TRAN[0], B_WAIT.
- Based on whether chip selects are used and connected to the logic analyzer, select **Connected** or **Unconnected** for each one, and indicate the base address to which the chip select refers.
- Indicate whether memory is Little endian or Big endian, or whether the BIGEND signal is being used.
- Indicate whether the B_WAIT signal is active high.
- For AMBA9 systems, the ARM equivalent, nEXEC signal is INSTREXEC.

Notes on the Signals Dialog

Chip Selects. This portion of the Signals Dialog provides information about chip select signals if they are available. The chip select signals allow the inverse assembler to regenerate upper address lines that are not connected. The logic analyzer expects the chip select lines to be *active low*.

Endian Selection. This option allows you to select between *little endian* and *big endian* memory configurations. Systems that switch between *little endian* and *big endian* for different memory devices must use the BIGEND signal to differentiate between these cases.

Wait Signals. For the ARM Core inverse assembler, the nWAIT signal is specified by ARM to be *active low*. However, some target systems use an *active high* wait signal. If your target has an *active high* wait signal, then nWAIT is being inverted. For this reason, select **Yes** in the “Is nWAIT inverted?” field.

For the AMBA ASB inverse assembler, the wait signal B_WAIT is specified by ARM to be *active high*. If your target system has an *active low* wait signal, then select **No** in the “Is B_WAIT active high?” field. (The default value of this field is **Yes**.)

ABORT and DMA Signals. The inverse assembler can use two signals that do not appear in the Signals Dialog list: ABORT and DMA. These signals are Active High, and, if connected, will be recognized by the logic analyzer and used by the inverse assembler. If they are not connected, the logic analyzer inputs will remain at logic Low, and the analyzer/inverse assembler will not use them.

To set AMBA AHB memory map preferences and signal information

Memory Map Information

The Memory Map dialog allows you to configure the inverse assembler to match the type of data in each memory region.

A memory region is a logical block of memory with a homogeneous set of characteristics. A memory region does not equate to a physical memory device; a single physical memory device can contain several memory regions for code and data space.

Describe your target circuit memory regions as follows:

- For up to 8 memory regions in your target system, indicate the Base Address, End Address, and Type of instruction.
- Types of memory accesses are: Inst. ARM, Inst. THUMB, and Data if HPROT0 is set to Unconnected.
- The inverse assembler assumes all memory regions are valid, so lower-numbered regions should be used before higher-numbered regions. Region 0 has the highest priority, and Region 7 has the lowest priority.
- If the inverse assembler returns “IA Error: Address not in map” then the address did not meet the specifications for any of the memory regions.

Signal Information

HPROT0. Decides whether the state is an opcode or data. When HPROT0 is set to **Connected**, the inverse assembler uses the HPROT0 signal to determine the state type. When HPROT0 is set to **Unconnected**, the state information is taken from the **Type** column in the **Memory Map Information**.

HWDATA. Single Master: Determines the source of the write data. When HWDATA is set to **Connected** the inverse assembler gets write data from HWDATA bus. When HWDATA is set to **Unconnected**, the inverse assembler gets write data from the HRDATA bus. Multiple Masters: The HWDATA selection has no effect.

HRESP1. Set to **Connected** to use the HRESP1 signal for the response type. If using the AHB-Lite bus specification, set to **unconnected** because the HRESP1 signal is not used.

Setting the Inverse Assembler Preferences

Endian Mode. Selects little endian or big endian mode.

Connector Pinout. Select whether the target is designed for single master or multiple master configuration. See “Single Master and Multiple Master Configurations” on page 90.

AMBA AHB Preferences Dialog

Invasm Preferences – AMBA AHB Data

AMBA AHB (E2493A) Preferences
Frame 10:Slot C:AMBA AHB Analyzer

Memory Map Information

Memory Region	Base Address	End Address	Type
Region 0	00000000	00000000	Inst. ARM ▾
Region 1	00000000	00000000	Inst. ARM ▾
Region 2	00000000	00000000	Inst. ARM ▾
Region 3	00000000	00000000	Inst. ARM ▾
Region 4	00000000	00000000	Inst. ARM ▾
Region 5	00000000	00000000	Inst. ARM ▾
Region 6	00000000	00000000	Inst. ARM ▾
Region 7	00000000	FFFFFFFF	Inst. ARM ▾

Signal Information

AMBA AHB Signals

HPR0T0 Connected ▾

HWDATA Connected ▾

HRESP1 Connected ▾

Other Information

Data Buses are little endian ▾

Connector Pinout: single master ▾

Apply Reset Close

Symbols

Symbols are more easily recognized than hexadecimal address values in logic analyzer trace displays, and they are easier to remember when setting up triggers.

HP logic analyzers let you assign user-defined symbol names to particular label values.

Also, you can download symbols from certain object file formats into HP logic analyzers.

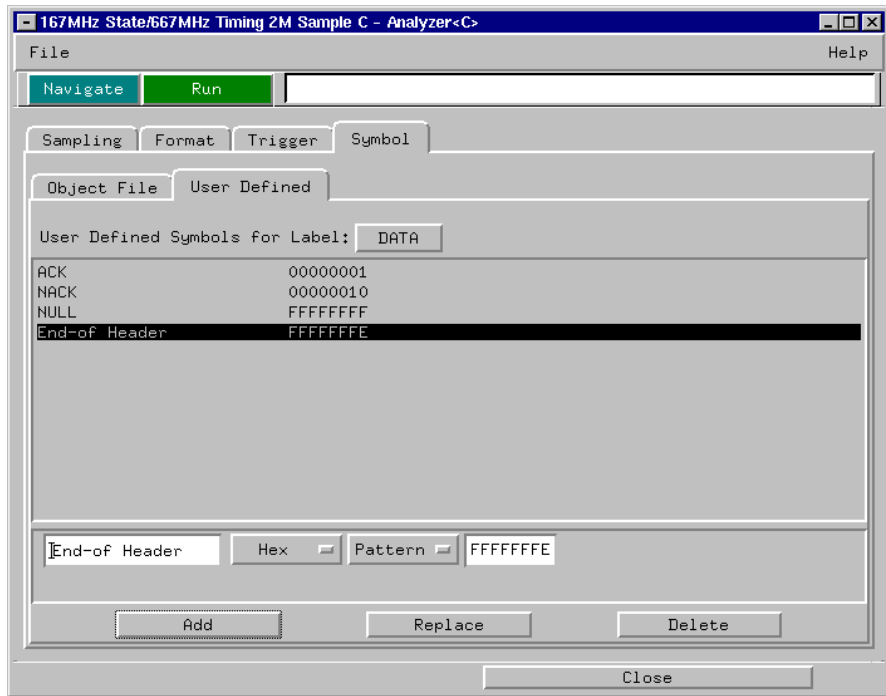
When source file line number symbols are downloaded to the logic analyzer, you can set up triggers on source lines. The HP B4620B Source Correlation Tool Set also lets you display the high-level source code associated with captured data.

User-defined symbols are symbols you create from within the logic analyzer user interface by assigning symbol names to label values. Typically, you assign symbol names to address label values, but you can define symbols for data, status, or other label values as well.

The User-Defined Symbols Dialog is shown in the next figure. Use this dialog to create your own symbols.

User-defined symbols are saved with the logic analyzer configuration.

User-Defined Symbols Dialog



Predefined ARM Symbols

The logic analyzer configuration files include predefined symbols.

These symbols appear along with the other user-defined symbols in the logic analyzer.

Object File Symbols

The most common way to load program symbols into the logic analyzer is from an object file that is created when the program is compiled.

Requirements

In order for object file symbols and source code to be accurately assigned to address values captured by the logic analyzer, you need:

An accurate bus trace

An Agilent Technologies logic analyzer is used to capture the microcontroller data.

An inverse assembler

The ARM inverse assembler decodes captured data into program counter (PC) addresses (also known as software addresses) and assembly language mnemonics.

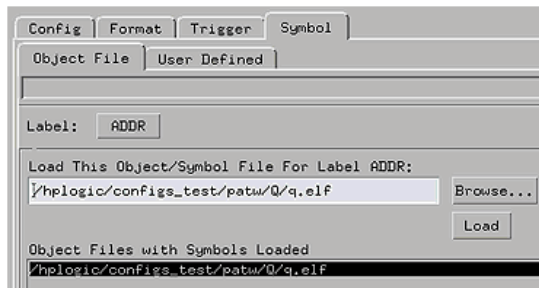
A symbol file

You need an object file containing symbolic debug information in a format the logic analyzer understands.

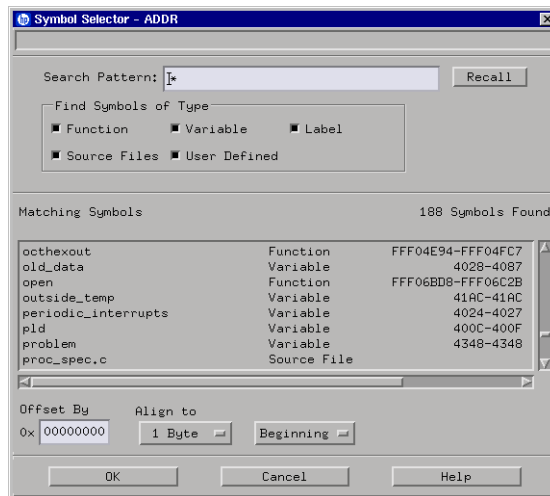
Alternatively, you can use a General Purpose ASCII (GPA) symbol file (see Chapter 11, “General-Purpose ASCII (GPA) Symbol File Format,” beginning on page 251).

To use object file symbols in the 16700

To load symbols in the Agilent 16700-series logic analysis system, open the logic analyzer module's Setup window and select the Symbol tab; then, select the Object File tab. Make sure the label is ADDR. From this dialog you can select object files and load their symbol information.



When you load object file symbols into a logic analyzer, a database of symbol/line number to address assignments is generated from the object file. The Symbol Selector dialog allows you to use a symbol in place of a hexadecimal value when defining trigger patterns, trigger ranges, and so on.



If your compiler generates files in a format that the logic analyzer doesn't understand, you can use a General-Purpose ASCII (GPA) symbol file. See Chapter 11, "General-Purpose ASCII (GPA) Symbol File Format," on page 251.

See Also

If you have an Agilent 16700-series logic analysis system, see the online help for more information on how to load symbols.

If you have another logic analyzer refer to your logic analyzer documentation for information on how to load symbol files.

Object File Symbols

**Configuring the
1660A/1670A/16500B/C-Series
Logic Analyzer**

Analyzing the ARM7/ARM9 with a 1660/1670/16500B/C Logic Analyzer

The sections of this chapter describe setting up and using the ARM inverse assembler. Because your ARM target system is designed uniquely according to your needs, it is important that you specify the available signals and memory regions to the inverse assembler.

NOTE:

The 1660/1670/16500B/C logic analyzers do not support AMBA AHB.

Configuring the Inverse Assembler

- See “Configuring Logic Analyzer IA Menus” on page 172 to set up inverse assembler preferences on a standalone logic analyzer.
-

Making Data Measurements

- See Chapter 8, “Capturing Processor Execution,” beginning on page 185 to instruct the logic analyzer to store the data you are interested in.
 - See Chapter 9, “Displaying Captured Processor Execution,” beginning on page 215 to interpret the inverse assembly results.
 - See “Making Common Measurements Using the Agilent 16700 Logic Analysis System” on page 197 or “Making Common Measurements Using All Other HP/Agilent Logic Analyzers” on page 205 for examples of common measurements.
-

Analyzer Modes

The logic analyzer can be configured to capture target system data in either state or timing mode.

State Mode

In state mode, the logic analyzer uses the MCLK (or BCLK) signal from the ARM target system to capture data synchronously. This mode allows inverse assembly of ARM instructions and is the default mode set up by the configuration files.

Timing Mode

In timing mode, the logic analyzer samples the incoming signals asynchronously, typically with 4 ns resolution. *Inverse assembly is not available in timing mode.*

The analyzer mode is set in the logic analyzer configuration menu.

Configuring Logic Analyzer IA Menus

Within the ARM inverse assembler on your logic analyzer, three menus allow you to set up your preferences and the specifics of your target circuit implementation. Visit each of these menus before running a trace.

- **Signals Menu** - Tells the inverse assembler about the availability and polarity of certain ARM/AMBA ASB signals. Based upon the signals that are connected in this menu, the inverse assembler will choose the algorithm that will produce the best inverse assembly output.
- **Memory Map Menu** - Tells the inverse assembler about data bus widths and the type of data in each memory region. The inverse assembler will use the values in the memory map to reconstruct missing status signals.
- **Filter Menu** - Tells the inverse assembler to show or suppress certain memory states.

After configuring your preferences in these menus, proceed to set up a trigger and run a data trace.

Signals Menu

Use the Signals Menu to tell the inverse assembler which target signals are available and connected to the logic analyzer.

For an ARM system:

ARM7TDMI Core Signals	Chip Select Signals	Base Address
nOPC <input type="button" value="Connected"/>	CS 0 <input type="button" value="Connected"/>	<input type="text" value="00010000"/>
MAS[1] <input type="button" value="Connected"/>	CS 1 <input type="button" value="Connected"/>	<input type="text" value="00020000"/>
MAS[0] <input type="button" value="Unconnected"/>	CS 2 <input type="button" value="Connected"/>	<input type="text" value="00030000"/>
nRW <input type="button" value="Unconnected"/>	CS 3 <input type="button" value="Unconnected"/>	<input type="text" value="00000000"/>
nEXEC <input type="button" value="Unconnected"/>	CS 4 <input type="button" value="Unconnected"/>	<input type="text" value="00000000"/>
DBGACK <input type="button" value="Connected"/>	CS 5 <input type="button" value="Unconnected"/>	<input type="text" value="00000000"/>
nTRANS <input type="button" value="Unconnected"/>	CS 6 <input type="button" value="Unconnected"/>	<input type="text" value="00000000"/>
nMREQ <input type="button" value="Connected"/>	CS 7 <input type="button" value="Unconnected"/>	<input type="text" value="00000000"/>
SEQ <input type="button" value="Unconnected"/>	Other Information	
nWAIT <input type="button" value="Connected"/>	Memory is <input type="button" value="Little"/> endian	
	Is nWAIT inverted? <input type="button" value="No"/>	

- Based on whether these signals are connected to the logic analyzer, select Connected or Unconnected for: nOPC, MAS[1], MAS[0], nRW, nEXEC, DBGACK, nTRANS, nMREQ, SEQ, nWAIT.
- Based on whether chip selects are used and connected to the logic analyzer, select Connected or Unconnected for each one, and indicate the base address to which the chip select refers.
- Indicate whether memory is Little endian or Big endian, or whether the BIGEND signal is being used.
- Indicate whether the nWAIT signal is inverted.

Chapter 7: Configuring the 1660A/1670A/16500B/C-Series Logic Analyzer

Configuring Logic Analyzer IA Menus

For an AMBA ASB system:

AMBA & ARM7 Core Signals	Chip Select Signals	Base Address
B_PROT[0] <input type="button" value="Connected"/>	CS 0 <input type="button" value="Connected"/>	00010000
B_SIZ[1] <input type="button" value="Connected"/>	CS 1 <input type="button" value="Connected"/>	00040000
B_SIZ[0] <input type="button" value="Unconnected"/>	CS 2 <input type="button" value="Unconnected"/>	00000000
B_WRITE <input type="button" value="Unconnected"/>	CS 3 <input type="button" value="Unconnected"/>	00000000
nEXEC <input type="button" value="Connected"/>	CS 4 <input type="button" value="Unconnected"/>	00000000
DBGACK <input type="button" value="Unconnected"/>	CS 5 <input type="button" value="Unconnected"/>	00000000
B_PROT[1] <input type="button" value="Connected"/>	CS 6 <input type="button" value="Unconnected"/>	00000000
B_TRAN[1] <input type="button" value="Connected"/>	CS 7 <input type="button" value="Unconnected"/>	00000000
B_TRAN[0] <input type="button" value="Unconnected"/>		
B_WAIT <input type="button" value="Connected"/>		

Other Information

Memory is endian

Is B_WAIT active high?

- Based on whether these signals are connected to the logic analyzer, select Connected or Unconnected for: B_PROT[0], B_SIZ[1], B_SIZ[0], B_WRITE, nEXEC, DBGACK, B_PROT[1], B_TRAN[1], B_TRAN[0], B_WAIT.
- Based on whether chip selects are used and connected to the logic analyzer, select Connected or Unconnected for each one, and indicate the base address to which the chip select refers.
- Indicate whether memory is Little endian or Big endian, or whether the BIGEND signal is being used.
- Indicate whether the B_WAIT signal is active high.
- For AMBA9 systems, the ARM equivalent, nEXEC signal is INSTREXEC.

Notes on the Signals Menu

Chip Selects. This portion of the Signals Menu provides information about chip select signals if they are available. The chip select signals allow the inverse assembler to regenerate upper address lines that are not connected. The logic analyzer expects the chip select lines to be *active low*.

On a 16500 or portable logic analyzer, the value of the chip select base address will be added to the current address. The resulting address will be compared to the addresses in the memory map and the appropriate region will be selected. Source line referencing is not available in this case.

Endian Selection. This option allows you to select between *little endian* and *big endian* memory configurations. Systems that switch between *little endian* and *big endian* for different memory devices must use the BIGEND signal to differentiate between these cases.

Wait Signals. For the ARM Core inverse assembler, the nWAIT signal is specified by ARM to be *active low*. However, some target systems use an *active high* wait signal. If your target has an *active high* wait signal, then nWAIT is being inverted. For this reason, enter ‘Yes’ in the “Is nWAIT inverted?” field.

For the AMBA ASB inverse assembler, the wait signal B_WAIT is specified by ARM to be *active high*. If your target system has an *active low* wait signal, then enter ‘No’ in the “Is B_WAIT active high?” field. (The default value of this field is ‘Yes’.)

ABORT and DMA Signals. The inverse assembler can use two signals that do not appear in the Signals Menu list: ABORT and DMA. These signals are Active High, and, if connected, will be recognized by the logic analyzer and used by the inverse assembler. If they are not connected, the logic analyzer inputs will remain at logic Low, and the analyzer/inverse assembler will not use them.

Memory Map Menu

The Memory Map menu allows you to configure the inverse assembler to match the ARM memory configuration regarding bus widths and the type of data in each memory region.

A memory region is a logical block of memory with a homogeneous set of characteristics. A memory region does not equate to a physical memory device; a single physical memory device can contain several memory regions for code and data space.

An ARM system may not always provide signals to distinguish instruction reads from data reads or ARM instructions from Thumb instructions. Also, there are no standard signals for specifying the width of the memory bus. When decoding a given address, the ARM inverse assembler uses the mapping information in the Memory Map menu to determine the access type and memory bus width.

	Base Address	End Address	Width	Type
Memory Region 0	00000000	00003FFF	32 Bits	Inst. ARM
Memory Region 1	00004000	00007FFF	16 Bits	Data
Memory Region 2	00008000	0000BFFF	8 Bits	Data
Memory Region 3	0000C000	0001FFFF	16 Bits	Inst. THUMB
Memory Region 4	00020000	0003FFFF	32 Bits	Data
Memory Region 5	00000000	00000000	32 Bits	Inst. THUMB
Memory Region 6	00000000	00000000	32 Bits	Inst. THUMB
Memory Region 7	00000000	00000000	32 Bits	Inst. THUMB

Describe your target circuit memory regions as follows:

- For up to 8 memory regions in your target system, indicate the Base Address, End Address, data bus Width and Type of instruction.
- Types of memory accesses are: Inst. ARM, Inst. THUMB, and Data.
- If nOPC and MAS[1] are supplied by your system, the Type field is not required to determine the instruction type and is ignored.

NOTE:

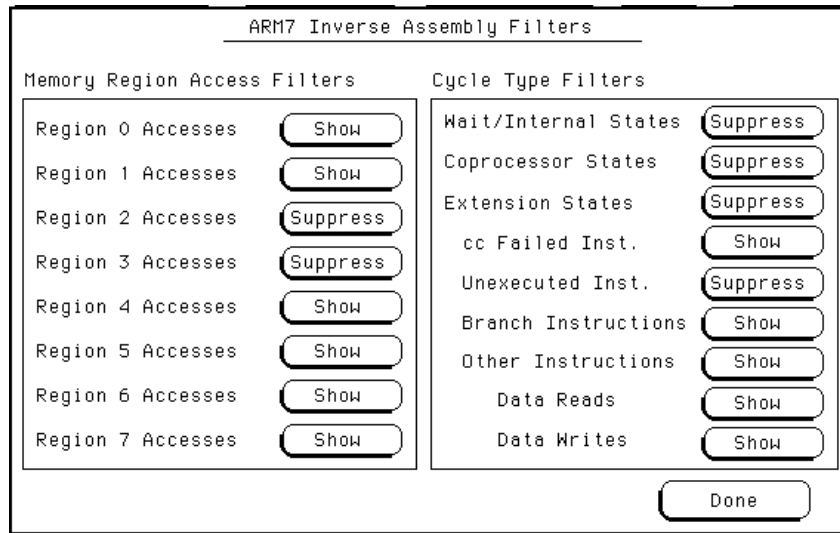
Regardless of the number of status bits available, the memory Width must always be specified.

The inverse assembler assumes all memory regions are valid, so lower-numbered regions should be used before higher-numbered regions. Region 0 has the highest priority, and Region 7 has the lowest priority.

If the inverse assembler returns "IA Error: Address not in map" then the address did not meet the specifications for any of the memory regions.

Filter Menu

The inverse assembler allows you to show or suppress several types of states using a function called *display filtering*. States can be filtered according to their cycle type (e.g. wait or read) or according to which memory region was accessed for the cycle.



The Show/Suppress settings do not affect the data that is stored by the logic analyzer, only whether the data is displayed. For different analysis requirements, the same data can be examined with different settings.

Display filtering allows faster analysis in two ways:

- 1 Unneeded information can be filtered out of the display. For example, suppressing Wait/Internal states will show only states in which an instruction or data fetch appears.
- 2 Particular operations can be isolated by suppressing all others. For example, quick analysis of branches can be shown by suppressing all other states.

- Show or Suppress each of the memory regions configured in the Memory Map menu.
- Show or Suppress other instructions based on their type:
 - Wait/Internal States - Wait states can be filtered with or without nWAIT. The inverse assembler will not mark any states as internal cycles without both nWAIT and nMREQ.
 - Coprocessor States - In order for these states to be filtered, nMREQ and SEQ must be connected. A coprocessor state is defined when nMREQ = 1 and SEQ = 1.
 - Extension states - States that have been used to build up instructions or data fetches in systems with 8 or 16-bit data buses.
 - cc Failed Inst. - These are instructions that have a condition code that did not pass. These instructions are marked with a '-'. nEXEC is required to mark any states as "cc Failed Inst".
 - Unexecuted Inst. - These are instructions that were not executed due to a pipeline flush. These instructions are marked with a '*'. nOPC is required to mark any state as "Unexecuted Inst.".
 - Branch Instructions - These can be any instruction that alters the flow of a program. Examples are loads to the PC, conditional branches, moves to the PC, software interrupts, etc.
 - Other Instructions - This category is for all other instructions that do not fall into any of the above categories.
 - Data Reads/Writes - These states include DMA reads/writes as well as normal core reads/writes.

Chapter 7: Configuring the 1660A/1670A/16500B/C-Series Logic Analyzer

Configuring Logic Analyzer IA Menus

Memory region filters have precedence over Cycle type filters.

The Agilent 16700-series logic analysis systems provide one additional feature for analyzer data. In addition to showing or suppressing states, the selected states can also be shown in color. Color can be used for either cycle types or memory regions, but not both at the same time

NOTE:

Color can be used for distinguishing either memory region accesses or cycle types, but not both at the same time.

NOTE:

For proper operation of the software analyzer (SWA), all unexecuted instructions due to pipeline flush should be filtered out.

Configuring the Logic Analysis System

You configure the logic analyzer by loading a configuration file. The information in the configuration file includes:

- Label names and channel assignments for the logic analyzer
- Inverse assembler file name

The configuration file you use is determined by the logic analyzer you are using, and whether you are performing state or timing analysis.

The procedures for loading a configuration file depend on the type of logic analyzer you are using. There is one procedure for the Agilent 16600/16700 series logic analysis system, and another procedure for the HP 1660-series, HP 1670-series, and logic analyzer modules in an HP 16500B/C mainframe. Use the appropriate procedures for your analyzer.

To load configuration and inverse assembler files— Agilent 16700 logic analysis systems

If you did not use Setup Assistant, you can load the configuration and inverse assembler files from the logic analysis system hard disk.

- 1** Click on the File Manager icon. Use File Manager to ensure that the subdirectory `/hplogic/configs/hp` exists.

If the above directory does not exist, you need to install the ARM Package. Close File Manager, then use the procedure on the CD-ROM jacket to install the ARM Package before you continue.

- 2** Using File Manager, select the configuration file you want to load in the `/hplogic/configs/hp` directory, then select Load. If you have more than one logic analyzer installed in your logic analysis system, use the Target field to select the machine you want to load.

The logic analyzer is configured for ARM analysis by loading the appropriate configuration file. Loading the indicated state file also automatically loads the inverse assembler. The configuration file you use is determined by the logic analyzer you are using, and whether you are performing state analysis or timing analysis.

- 3** Close File Manager.

To load configuration files—other logic analyzers

If you have an HP 1660-series, HP 1670-series, or logic analyzer modules in an HP 16500B/C mainframe use these procedures to load the configuration file and inverse assembler.

The first time you set up the logic analyzer, make a duplicate copy of the flexible disk. For information on duplicating disks, refer to the reference manual for your logic analyzer.

For logic analyzers that have a hard disk, you might want to create a directory such as ARM on the hard drive and copy the contents of the floppy onto the hard drive. You can then use the hard drive for loading files.

Configuring the logic analyzer consists of loading the software by inserting the floppy disk into the logic analyzer disk drive and loading the proper configuration file.

- 1** Insert the floppy disk in the front disk drive of the logic analyzer.
- 2** Go to the Flexible Disk menu.
- 3** Configure the menu to load.
- 4** Use the knob to select the appropriate configuration file.

The configuration file you use is determined by the logic analyzer you are using, and whether you are performing state analysis or timing analysis.

- 5** Select the appropriate analyzer on the menu.
- 6** Execute the load operation on the menu to load the file into the logic analyzer.

The logic analyzer is configured for ARM analysis by loading the appropriate configuration file. Loading a state configuration file also automatically loads the inverse assembler.

Chapter 7: Configuring the 1660A/1670A/16500B/C-Series Logic Analyzer
Configuring the Logic Analysis System

Capturing Processor Execution

Chapter 8: Capturing Processor Execution

The normal steps in using the logic analyzer are:

1. Configure the logic analyzer.
2. Format labels for the logic analyzer channels (that is, mapping logic analyzer channels to target system signal names).
3. Load symbols from the program's object file.
4. Set up the trigger, and run the measurement.
5. Display the captured data.

The logic analyzer is configured and labels are created (formatted) for the logic analysis channels when configuration files are loaded. See “Configuring the 16700-series Logic Analysis System” on page 141 or “Configuring the 1660A/1670A/16500B/C-Series Logic Analyzer” on page 169.

You can load program object file symbols into the logic analyzer when configuring it. See “Object File Symbols” on page 165.

This chapter describes setting up logic analyzer triggers when using the inverse assembler and B4620B source correlation tool set.

See Chapter 9, “Displaying Captured Processor Execution,” beginning on page 215 for information on displaying captured data.

Trigger sequence

The Trigger sequence is set up by the software to store all states.

Triggering allows the logic analyzer to store only the data states that you want to see, ensuring quicker analysis of the stored data.

NOTE:

If you modify the trigger sequence to store only selected bus cycles, incorrect or incomplete disassembly may be displayed.

Some systems may be limited in their ability to trigger on data values or certain opcodes due to smaller data bus widths. For example, a system with an 8-bit data bus will need at least a four level trigger sequence to trigger on an ARM opcode.

Predefined trigger terms

Included in the logic analyzer configuration files are several predefined trigger terms to simplify trigger setup.

The terms are:

- i fetch - Instruction fetch (Requires nOPC)
- d fetch - Data fetch (Requires nOPC)
- d read - Data read (Requires nOPC and nRW)
- d write - Data write (Requires nOPC and nRW)
- address
- data
- debug (Requires DBGACK)
- wait (Requires nWAIT)

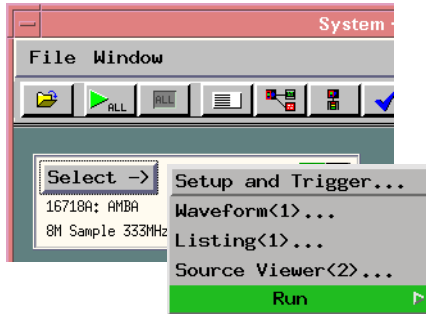
NOTE:

In order for these trigger terms to work correctly, the associated status signals listed above must be connected and an ARM configuration file must be loaded.

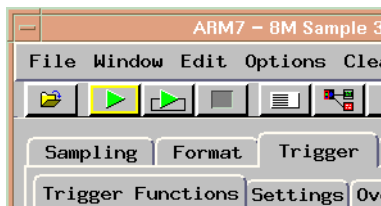
Predefined trigger terms**To use predefined trigger terms****NOTE:**

A configuration file for an ARM processor must be loaded. You can load an ARM configuration file using the Setup Assistant (see page 22).

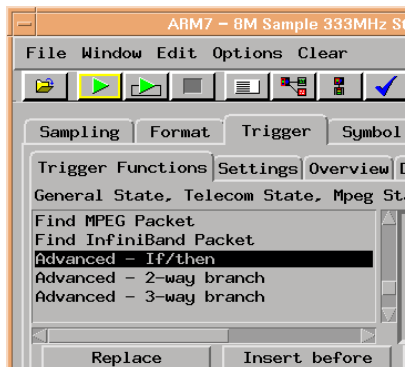
- 1 From the System window, select the **Select** tab for the logic analyzer. Select **Setup and Trigger...** from the menu that appears.



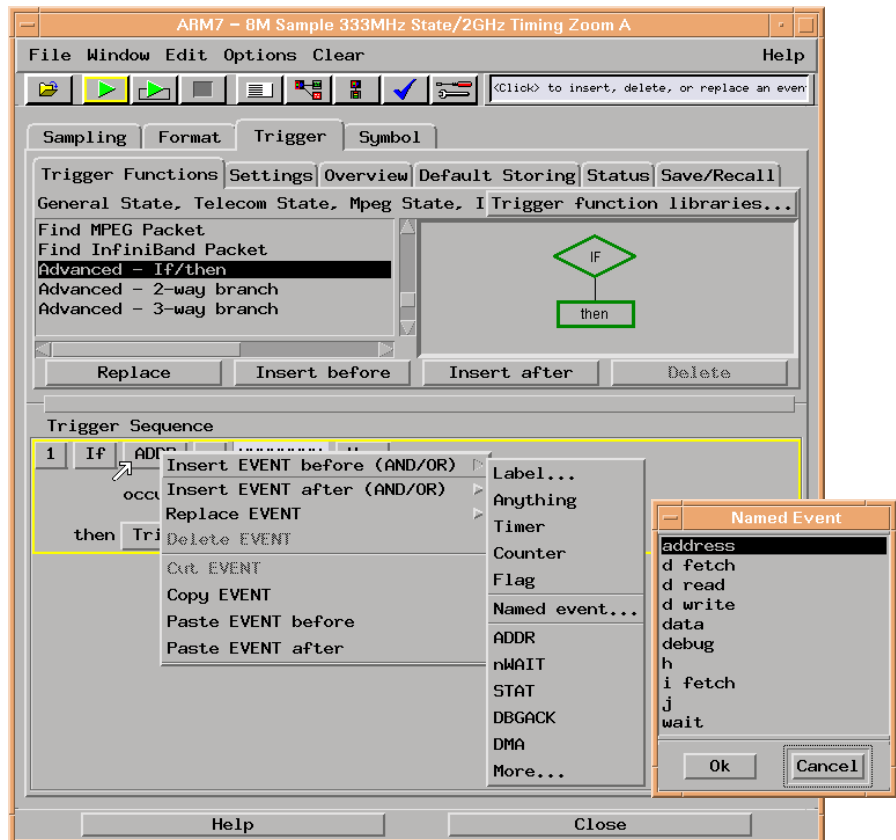
- 2 Select the logic analyzer's **Trigger** tab, then select the **Trigger Functions** tab.



- 3 Scroll down and select **Advanced If/Then**, and then select the **Replace** button.



- 4 In the Trigger Sequence box, select the ADDR button.
- 5 Select Insert EVENT before (AND/OR).
- 6 Select Named event.... The Named Event dialog box will appear.

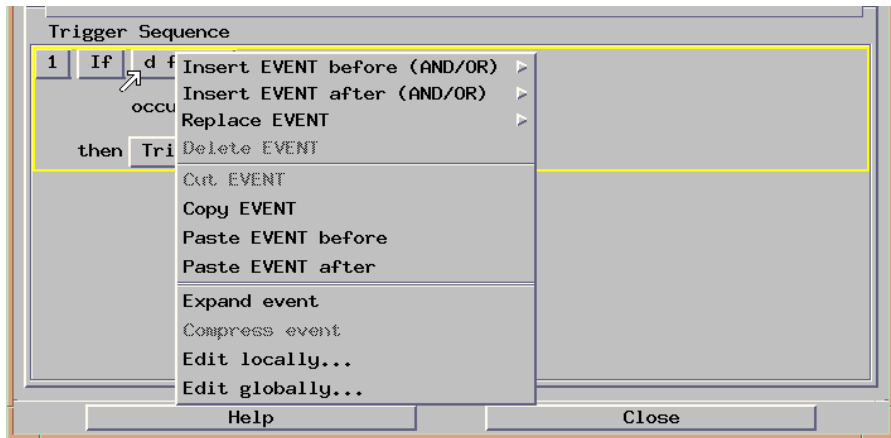


- 7 Choose a predefined trigger term.

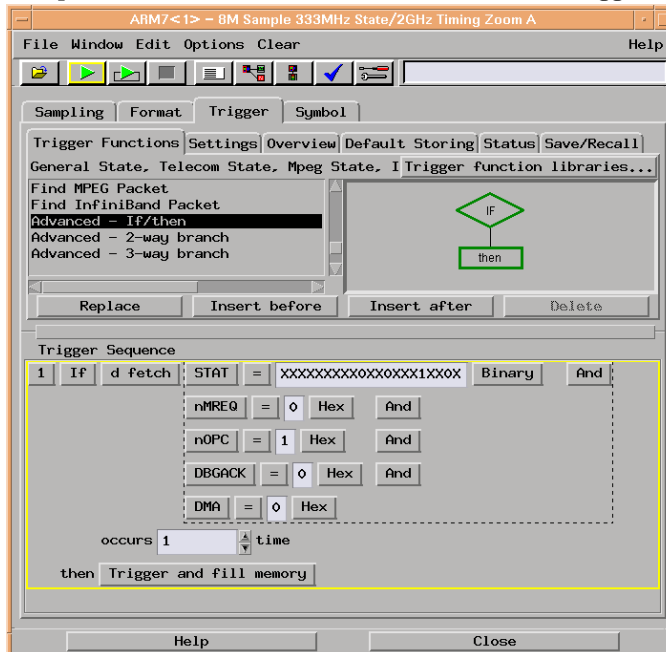
Predefined trigger terms

To view the definition of the trigger term

From the **Trigger** tab, select the named event (the cursor is pointing to it in the picture below). A menu will appear. Select **Expand event** from the menu.

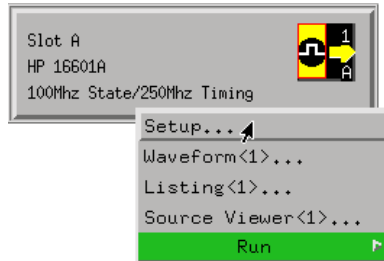


The event will be expanded, and the terms of the trigger event will be shown. The picture below shows the terms of the **d fetch** trigger event.



To Set Up Logic Analyzer Triggers

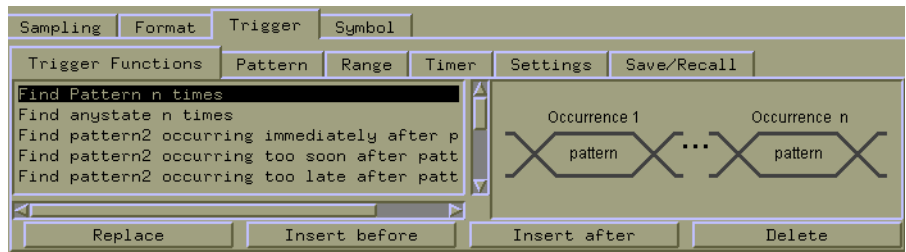
- 1 Open the logic analyzer's Setup window.



- 2 Select the Trigger tab.

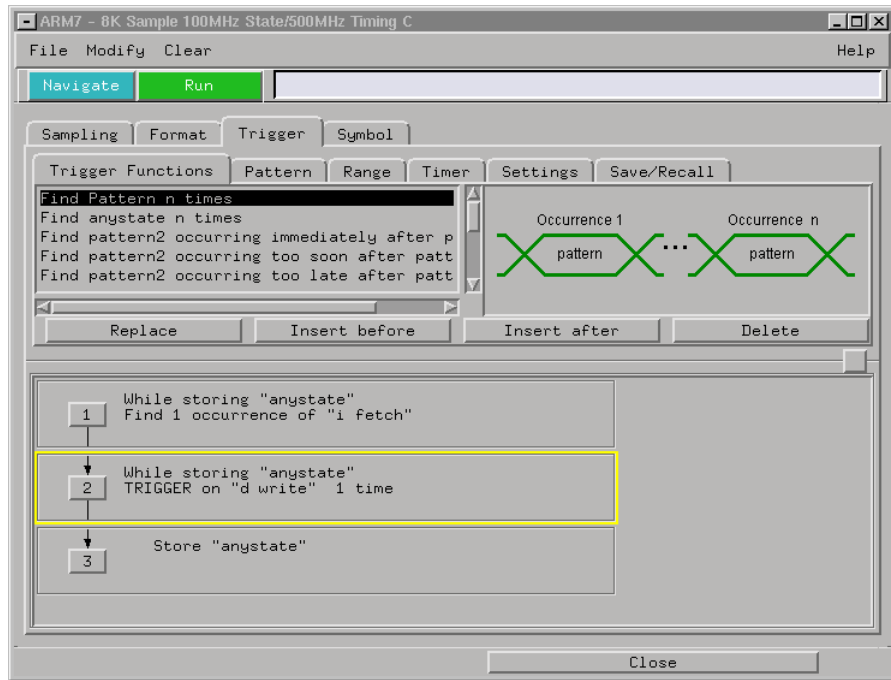


- 3 Select the trigger function that will be used in the logic analysis measurement.



Chapter 8: Capturing Processor Execution To Set Up Logic Analyzer Triggers

4 Set up the trigger sequence.



5 Run the measurement.



See Also

See the Agilent 16700-series logic analysis system's on-line help for more information on setting up logic analyzer triggers.

Triggering on Symbols and Source Code

When setting up trigger specifications to capture processor execution:

- Use the logic analyzer trigger alignment to avoid missed triggers.
- Use the logic analyzer address offset to compensate for relocated code.
- Use the logic analyzer storage qualification to capture the software execution you're interested in and filter out library code execution (whose source file lookups can take a long time if the library source code is not available).

To avoid triggering on prefetched instructions

An ARM microcontroller may prefetch two instructions following a taken branch. The inverse assembler does not filter these prefetches. This means that the prefetched states will be collected by the analyzer, and that a trigger set to the address of the prefetched instruction will cause a false trigger on the prefetch.

The recommended way to avoid false triggering for an ARM device (32-bit data bus) in this case is to offset the address of the trigger by 8. An offset field is provided in the symbolic trigger menu to allow offsetting the symbol address.

Note that this is not a foolproof scheme, since this may result in an inappropriate trigger if the offset address is a point where control transfers (branch destination). Be aware of prefetches and adjust your triggering to compensate for it as you gain experience with the processor and your code.

Example

A common example of this is setting a trigger on the source line following a loop, for instance:

Line #	Addr	C source	Assembly Source
100	1000	for (i=0;i<10;i++)	
	1000		MOVEQ #0,D3
101	1002	{	
	1002		forLoop1:
102	1002	foo = foo + 100;	
	1002		ADD.L #100,D2
103	1008	}	
	1008		ADDQ.L #1,D3
	100A		CMP.L #10,D3
	1010		BLT forLoop1
104	1012	printf("%d\n", foo);	
	1012		-MOVEA.L D2,A0

The instruction at 1012 will be prefetched following the BLT at address 1010. So, setting a trigger on line #104 (address 1012) will result in a premature trigger.

To correlate relocatable code using the address offset

You need to adjust the source correlation tool set to compensate for relocatable code segments or memory management units that produce fixed code offsets.

The offset field in the trigger menu allows you to offset the symbol address. Entering the appropriate address offset will cause the source correlation tool set to reference the correct symbol information for the relocatable or offset code.

To adjust for prefetches, use a trigger offset of 8 (prefetch queue depth) to avoid triggering on prefetched instructions. This is not a foolproof scheme, since this may result in a missed trigger if a branch takes place between the base address and the offset address. For the ARM, an offset of 8 is large enough to overcome the prefetch queue.

Triggering ARM Data on the 1660/70-series logic analyzers

Triggering allows the logic analyzer to store only the data states that you want to see, ensuring quicker analysis of the stored data.

Some systems may be limited in their ability to trigger on data values or certain opcodes due to smaller data bus widths. For example, a system with an 8-bit data bus will need at least a four level trigger sequence to trigger on an ARM opcode.

A sample logic analyzer trigger window is shown here:

1M Sample LA E
Trigger 1
Print
Run

State Sequence Levels

1

While storing "≠DEBUG"
 TRIGGER on "address" occurring 2 times

Timer
1 2
- -

Arming Control

2

Store "≠d fetch≠WAIT"

- -

Acquisition Control

Count Off

Modify Trigger

◀Label▶	nMREQ	nEXEC	nRW	nOPC	nTRANS	MAS1:0
↑Terms↓	Hex	Hex	Hex	Hex	Hex	Hex
i fetch	0	X	X	0	X	X
d fetch	0	X	X	1	X	X
d read	X	X	0	1	X	X
d write	X	X	1	1	X	X

Chapter 8: Capturing Processor Execution

Triggering on Symbols and Source Code

Included in the logic analyzer configuration files are several predefined trigger terms to simplify trigger setup.

The terms are:

- i fetch - Instruction fetch (Requires nOPC)
- d fetch - Data fetch (Requires nOPC)
- d read - Data read (Requires nOPC and nRW)
- d write - Data write (Requires nOPC and nRW)
- address
- data
- debug (Requires DBGACK)
- wait (Requires nWAIT)

NOTE:

In order for these trigger terms to work correctly, the associated status signals listed above must be connected.

Making Common Measurements Using the Agilent 16700 Logic Analysis System

This section provides several examples of common trigger setups for the Agilent 16700 logic analysis systems.

Example 1: Setting a trigger for a specific address

This trigger is useful when a particular function or variable may be causing a problem. By triggering on the address at the start of a function or on the address of a particular variable, debug time can be minimized significantly since the logic analyzer trace will contain only the trigger states that are related to that function or variable.

- 1** In the Trigger tab, select Clear, and then All from the menu bar.
- 2** Select trigger level 1, and then Edit...
- 3** Select TRIGGER on to 'address' then Close. Scroll down to the pattern 'address'.
- 4** Next to the ADDR label, enter the address you want to trigger n for the 'address' pattern. (You may have to “don't care” the two least significant address bits for the 32-bit data bus systems if they are not valid on your system.)
- 5** The trigger is complete. Select the Run button to begin looking for the trigger condition.

Example 2: Triggering on a write to a variable

If you suspect a problem when writing to a pointer variable, set up the logic analyzer to trigger on a write to the variable. For this example, assume that the address location is 0x08002400 and the value we are writing is 0x08002600.

For a 32-bit data bus system:

- 1** In the Trigger tab, select Clear, and then All from the menu bar.
- 2** Select trigger level 1, and then Edit... Since we want to view trace data before and after the trigger has occurred, set the While storing field to 'Anystate'.
- 3** Set the TRIGGER on field to 'd write' and then select Done.
- 4** Locate the "d write" term at the bottom of the Trigger menu, set the ADDR field to "08002400" and the DATA field to "08002600".

The trigger is complete. Select the Run button to begin looking for the trigger condition.

Example 3: Triggering on a 16-bit write to variable

Assume that there is a 4-byte counter variable at address 08002402 in a program that keeps a count of the number of times an operation has been performed. However, there seems to be a problem with the system after the counter reaches 0x05236400. In order to determine what happens after the counter reaches that number, a properly designed trigger is needed.

- 1** In the Trigger tab, select Clear, and then All from the menu bar.
- 2** Select trigger level 1, then Insert before...
- 3** Select the default trigger macro (User level custom combinations, loops), select OK, and then Close.
- 4** Select trigger level 1, then Edit...
- 5** In the While storing field, select 'no state'.
- 6** In the Find field, select 'd write'.
- 7** In the Else on field, select 'no state', then Done.
- 8** Select trigger level 2, then Edit...
- 9** In the While storing field, select 'anystate'.
- 10** In the TRIGGER on field, select 'Combination'.
In the combination window:
 - Select 'd fetch -> ON' and 'wait -> NEGATE'.
 - Select the operator between these two terms to be AND.
(See the screen shot on the following page)
 - The current qualifier should appear as "d fetch . !=wait". Select OK.
- 11** In the trigger level 2 Else on field, select 'No state', then Close.
- 12** Select trigger level 3, then Edit...
- 13** In the Store field, select 'anystate', then Close.

- 14** For the “d fetch” term, place 08002402 under the ADDR label and XXXX0523 under the DATA label.
- 15** For the “d write” term at the bottom of the menu, place 08002400 under the ADDR label and XXXX6400 under the DATA label.
- 16** The trigger is complete, select the Run button to begin looking for the trigger condition.

Example 3 Notes

This setup only works for little endian systems. For big endian systems, *d.write* and *d.fetch* will need to be swapped.

A few interesting things should be mentioned with this trigger setup. Since the data bus is only 16 bits wide, the memory controller naturally asserts the *nWAIT* line until all of the data for a 32-bit request has been received. For this reason, wait states must be taken into account when working with data bus sizes smaller than 32-bits. Since wait states aren't being ignored by the logic analyzer in this case, a wait state will most likely trigger the logic analyzer. In order to capture all of the useful data, the logic analyzer stores all of the data between the first two levels.

This creates the possibility of a false trigger since the logic analyzer is basing its trigger on 2 pieces of 16-bit data. For example, perhaps there were 3 writes to the variable location at 08002400. The three writes had values of 05216400, 05226400, and 05236400. Since 05216400 appeared on the bus first, the logic analyzer would have started storing data when the first write occurred. The logic analyzer will have stored all of the data starting with the first write of 05216400 and ending with the second half of the write 05236400.

An easier way of triggering on two or more data cycles would be to use the ARM debugger to set an internal hardware breakpoint on the data in question. Refer to the ARM debugger manuals for more information.

Example 4: Setting a trigger at exit of debug mode

This trigger is most useful when a debugger is being used in conjunction with an emulation probe/module. This setup prevents the logic analyzer from storing any data until the probe/module starts execution of the target processor. Without this trigger, the logic analyzer would begin storing states while the target processor is still in debug mode which would fill up the logic analyzer with useless data.

Note: DBGACK is required for this trigger to work correctly.

- 1** In the Trigger tab, select Clear, and then All from the menu bar.
- 2** Select trigger level 1, then Edit...
- 3** Set While storing to '!= DEBUG' and set TRIGGER on to '!= DEBUG'.
- 4** Select trigger level 2, then Edit...
- 5** Set Store to 'anystate'.

Example 5: Store qualifying wait states

This setup is useful for systems that have memory devices that require several wait states for each access. This measurement will only store non-wait states. This has the positive effect of saving logic analyzer memory for states that might be more useful for debugging.

Note: nWAIT/B_WAIT is required for correct operation, and can only be used with 32-bit data bus systems. (nWAIT is asserted during the extra cycles used to make up the full core memory bus in 8 or 16-bit data bus systems) Store qualifying on nWAIT/B_WAIT will prevent capturing needed data bus cycles. *(Use filtering instead for 8 or 16-bit data bus systems.)*

- 1** In the Trigger tab, select Clear, and then All from the menu bar.
- 2** Select trigger level 2, then Edit...
- 3** Set the Store field to '!= WAIT'.
- 4** An alternate method:
- 5** Go to the Format tab of logic analyzer.
- 6** Connect the nWAIT signal to an unused clock bit.
- 7** Go to the Master Clock menu.
- 8** If nWAIT is active-low, for the qualifier Q1, select 'clock letter' high.
- 9** If nWAIT is active-high, for the qualifier Q1, select 'clock letter' low.
- 10** Repeat for the Slave Clock menu.

Making Common Measurements Using All Other HP/Agilent Logic Analyzers

This section provides several examples of common trigger setups for the HP 16500 and HP/Agilent portable logic analyzers.

Example 1: Setting a trigger for a specific address

This trigger is useful when a particular function or variable may be causing a problem. By triggering on the address at the start of a function or on the address of a particular variable, debug time can be minimized significantly since the logic analyzer trace will contain only the trigger states that are related to that function or variable.

- 1 In the Trigger menu, clear any previous triggers by selecting 'Modify Trigger->Clear Trigger->Sequence Levels'.
- 2 Select trigger level 1.
- 3 Select TRIGGER on to 'address', then *Done*. Scroll down to the trigger term 'address'.
- 4 Under the ADDR label, enter the address you want to trigger on for the 'address' term. (You may have to "don't care" the two least significant address bits for 32-bit data bus systems if they are not valid on your system.)
- 5 The trigger is complete, press Run to begin looking for the trigger condition.

The screenshot shows the logic analyzer's Trigger menu. At the top, there are buttons for '1M Sample LA E', 'Trigger 1', 'Print', and 'Run'. Below these, the 'State Sequence Levels' section shows a sequence starting with level 1: 'While storing "no state" TRIGGER on "address" occurring 1 time', followed by level 2: 'Store "anystate"'. To the right of the sequence levels are buttons for 'Arming Control', 'Acquisition Control', 'Count Off', and 'Modify Trigger'. Below this is a table for configuring the trigger terms.

	B_CLK	ADDR	DATA	STAT	TRAI:0	nE
Terms	Hex	Hex	Hex	Binary	Hex	He
wait	X	XXXXXXXX	XXXXXXXX	X0XX1XXXXXXXXX	X	X
debug	X	XXXXXXXX	XXXXXXXX	XXXXX1XXXXXXXX	X	X
address	X	00008080	XXXXXXXX	XXXXXXXXXXXXXXXX	X	X
data	X	XXXXXXXX	XXXXXXXX	XXXXXXXXXXXXXXXX	X	X

Example 2: Triggering on a write to a variable

If you suspect a problem when writing to a pointer variable, set up the logic analyzer to trigger on a write to the variable. For this example, assume that the address location is 0x08002400 and the value we are writing is 0x08002600.

For a 32-bit data bus system:

- 1 In the Trigger menu, clear any previous triggers by selecting *'Modify Trigger -> Clear Trigger -> Sequence Levels'*.
- 2 Select trigger level 1. Since we want to view trace data before and after the trigger has occurred, set the While storing field to *'anystate'*.
- 3 Set the TRIGGER on field to *'d write'* and then select *Done*.
- 4 Locate the *"d write"* term at the bottom of the Trigger menu, set the ADDR field to *"08002400"* and the DATA field to *"08002600"*.
- 5 The trigger is complete, press Run to begin looking for the trigger condition.

Here is a Trigger menu containing the above setup:

IM Sample LA E
Trigger 1
Print
Run

State Sequence Levels

1 While storing "anystate"
TRIGGER on "d write" occurring 1 time

↓

2 Store "anystate"

Timer 1 2

Arming Control

Acquisition Control

Count Off

Modify Trigger

←Label→	MCLK	ADDR	DATA	STAT	SEQ	nMR
→Terms→	Hex	Hex	Hex	Binary	Hex	Hex
i fetch	X	XXXXXXXX	XXXXXXXX	XOX1OXXXOXXOX	X	0
d fetch	X	XXXXXXXX	XXXXXXXX	XOX1OXXX1XXOX	X	0
d read	X	XXXXXXXX	XXXXXXXX	XOX1OXXX1OXOX	X	0
d write	X	08002400	08002600	XOX1OXXX11XOX	X	0

Example 3: Triggering on a 16-bit write to variable

Assume that there is a 4-byte counter variable at address 08002402 in a program that keeps a count of the number of times an operation has been performed. However, there seems to be a problem with the system after the counter reaches 0x05236400. In order to determine what happens after the counter reaches that number, a properly designed trigger is needed.

- 1** In the Trigger menu, clear any previous triggers by selecting '*Modify Trigger -> Clear Trigger -> Sequence Levels*'.
- 2** Select trigger level 1, select TRIGGER on to '*anystate*', then '*Insert Level -> Before*'.
- 3** Select the default trigger macro (User level custom combinations, loops), then *Done*.
- 4** In the While storing field, select '*no state*'.
- 5** In the Find field, select '*d write*'.
- 6** In the Else on field, select '*no state*', then *Done*.
- 7** Select trigger level 2. In the While storing field, select '*anystate*'.
- 8** In the TRIGGER on field, select '*Combination*'.
In the combination window:
 - Select '*d fetch -> ON*' and '*wait -> NEGATE*'.
 - Select the operator between these two terms to be *AND*.
(See the screen shot on the following page)
 - The current qualifier should appear as "*d fetch . !=wait*". Select *Done*.
- 9** In the trigger level 2 Else on field, select '*no state*', then *Done*.
- 10** Select trigger level 3. In the Store field, select '*anystate*', then *Done*.
- 11** For the "d fetch" term, place *08002402* under the ADDR label and *XXXX0523* under the DATA label.

- 12** For the “d write” term at the bottom of the menu, place *08002400* under the ADDR label and *XXXX6400* under the DATA label.
- 13** The trigger is complete, press Run to begin looking for the trigger condition.

Here is a Trigger menu containing the above setup:

IM Sample LA E
Trigger 1
Print
Run

State Sequence Levels

1

While storing “no state”

Find “d write” occurring 1 time

↓

2

While storing “anystate”

TRIGGER on “d fetch+wait” occurring 1 time

↓

3

Store “anystate”

Timer

1 2

-- --

-- --

-- --

Arming Control

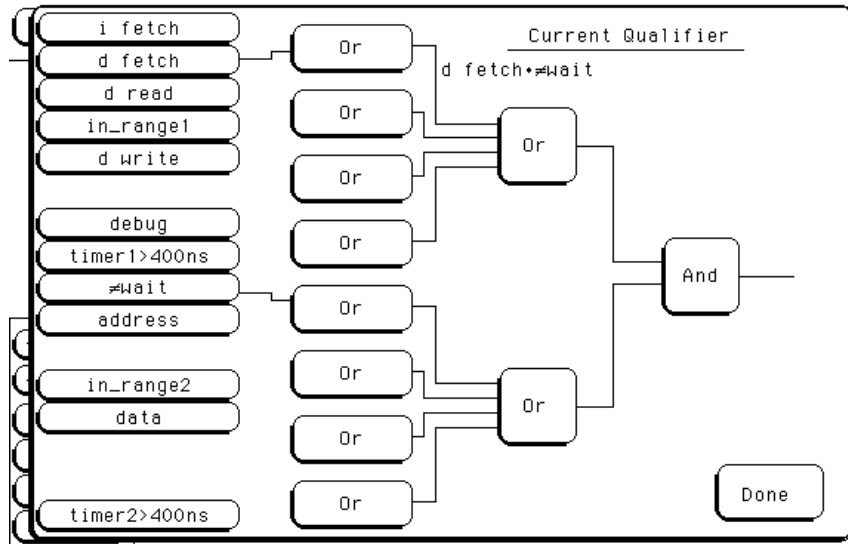
Acquisition Control

Count Off

Modify Trigger

←Label→	MCLK	ADDR	DATA	STAT	SEQ
↑Terms↓	Hex	Hex	Hex	Binary	Hex
i fetch	X	XXXXXXXX	XXXXXXXX	XXXXXXXXXX0XX0XX0XX0X	X
d fetch	X	08002402	XXXX0523	XXXXXXXXXX0XX0XXX1XX0X	X
d read	X	XXXXXXXX	XXXXXXXX	XXXXXXXXXX0XX0XXX10X0X	X
d write	X	08002400	XXXX6400	XXXXXXXXXX0XX0XXX11X0X	X

Here is the set up of terms in the Combination menu:



Example 3 Notes

This setup only works for little endian systems. For big endian systems, *d.write* and *d.fetch* will need to be swapped.

A few interesting things should be mentioned with this trigger setup. Since the data bus is only 16 bits wide, the memory controller naturally asserts the *nWAIT* line until all of the data for a 32-bit request has been received. For this reason, wait states must be taken into account when working with data bus sizes smaller than 32-bits. Since wait states aren't being ignored by the logic analyzer in this case, a wait state will most likely trigger the logic analyzer. In order to capture all of the useful data, the logic analyzer stores all of the data between the first two levels.

This creates the possibility of a false trigger since the logic analyzer is basing its trigger on 2 pieces of 16-bit data. For example, perhaps there were 3 writes to the variable location at 08002400. The three writes had values of 05216400, 05226400, and 05236400. Since 05216400 appeared on the bus first, the logic analyzer would have started storing data when the first write occurred. The logic analyzer will have stored all of the data starting with the first write of 05216400 and ending with the second half of the write 05236400.

An easier way of triggering on two or more data cycles would be to use the ARM debugger to set an internal hardware breakpoint on the data in question. Refer to the ARM debugger manuals for more information.

Example 4: Setting a trigger at exit of debug mode

This trigger is most useful when a debugger is being used in conjunction with an emulation probe/module. This setup prevents the logic analyzer from storing any data until the probe/module starts execution of the target processor. Without this trigger, the logic analyzer would begin storing states while the target processor is still in debug mode which would fill up the logic analyzer with useless data.

Note: DBGACK is required for this trigger to work correctly.

- 1 In the Trigger menu, clear any previous triggers by selecting 'Modify Trigger -> Clear Trigger -> Sequence Levels'.
- 2 Select trigger level 1, then While storing to '!= DEBUG'.
- 3 In trigger level 1, set TRIGGER on to '!= DEBUG'.
- 4 Select trigger level 2, then Store to 'anystate'.

Here is a Trigger menu containing the above setup:

Label	MCLK	ADDR	DATA	STAT	SEQ	nMR
Terms	Hex	Hex	Hex	Binary	Hex	Hex
i fetch	X	XXXXXXXX	XXXXXXXX	XOX10XXX0XXOX	X	0
d fetch	X	XXXXXXXX	XXXXXXXX	XOX10XXX1XXOX	X	0
d read	X	XXXXXXXX	XXXXXXXX	XOX10XXX10XOX	X	0
d write	X	XXXXXXXX	XXXXXXXX	XOX10XXX11XOX	X	0

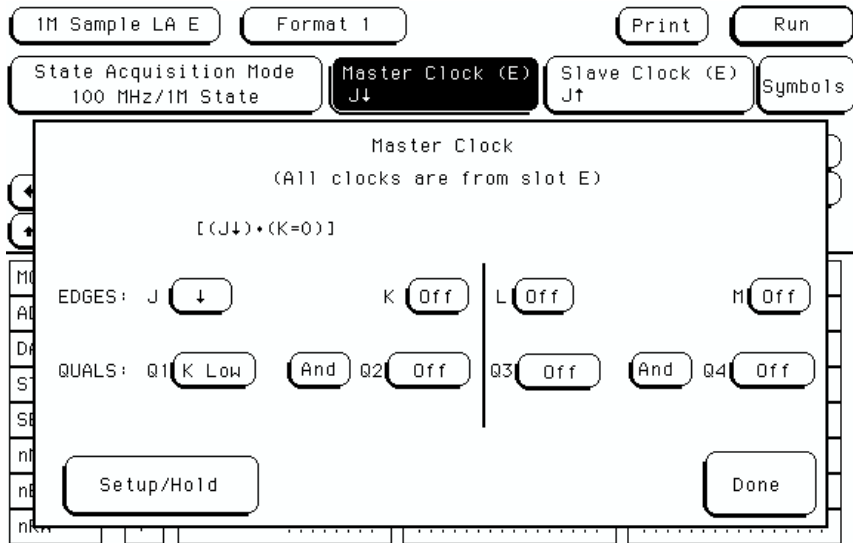
Example 5: Store qualifying wait states

This setup is useful for systems that have memory devices that require several wait states for each access. This measurement will only store non-wait states. This has the positive effect of saving logic analyzer memory for states that might be more useful for debugging.

Note: nWAIT/B_WAIT is required for correct operation, and can only be used with 32-bit data bus systems. (nWAIT is asserted during the extra cycles used to make up the full core memory bus in 8 or 16-bit data bus systems) Store qualifying on nWAIT/B_WAIT will prevent capturing needed data bus cycles. (*Use filtering instead for 8 or 16-bit data bus systems.*)

- 1** In the Trigger menu, clear any previous triggers by selecting '*Modify Trigger -> Clear Trigger -> Sequence Levels*'.
- 2** Select trigger level 2, then the Store field to '*!= WAIT*'.
- 3** An alternate method:
- 4** Go to the Format menu of logic analyzer.
- 5** Connect the nWAIT signal to an unused clock bit.
- 6** Go to the Master Clock menu.
- 7** If nWAIT is active-low, for the qualifier Q1, select '*clock letter*' *high*.
- 8** If nWAIT is active-high, for the qualifier Q1, select '*clock letter*' *low*.
- 9** Repeat for the Slave Clock menu.

This menu shows the results of the previous setup:



Displaying Captured Processor Execution

Viewing ARM Trace Data

This section discusses the general output format of the inverse assembler and processor-specific information.

To display captured state data:

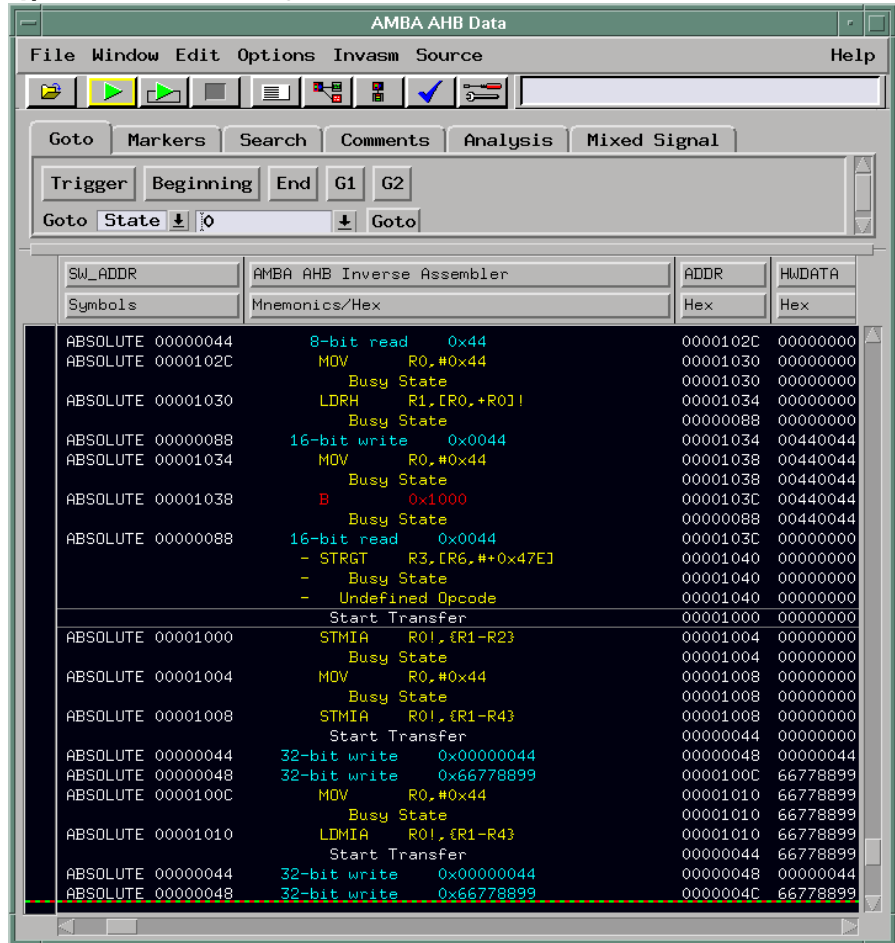
- If you are using an Agilent 16700-series logic analysis system, select the analyzer you are using, and then choose **Listing...** from the pop-up menu.
- For all other analyzers, select the listing menu
- Set the base for the DATA label to Invasm.

Typical AMBA ASB Listing

State Number	ADDR	ARM & Thumb Mnemonics v1.30	DATA
Decimal	Hex	Inverse Assembler	Hex
91	0B3A2	CMP R0 ,#0x00	2800
92	0B3A4	<wait>	2800
93	0B3A4	<wait>	D001
94	0B3A4	BEQ 0000B3AA	D001
95	0234C	<wait>	D001
96	0234C	<wait>	D001
97	0234C	<wait>	D030
98	0234C	READ BYTE 0x30	D030
99	0B3A6	<wait>	D030
100	0B3A6	<wait>	D030
101	0B3A6	<wait>	3101
102	0B3A6	ADD R1 ,#0x01	3101
103	0B3A8	<wait>	3101
104	0B3A8	<wait>	E7FA
105	0B3A8	B 0000B3A0	E7FA
106	0B3AA	<wait>	E7FA
107	0B3AA	<wait>	781F
108	0B3AA	* LDRB R7, [R3, #0x00]	781F
109	0B3AC	<wait>	781F
110	0B3AC	<wait>	3301
111	0B3AC	* ADD R3 ,#0x01	3301
112	0B3A0	<wait>	3301
113	0B3A0	<wait>	7808
114	0B3A0	LDRB R0, [R1, #0x00]	7808
115	0B3A2	<wait>	7808
116	0B3A2	<wait>	2800
117	0B3A2	CMP R0 ,#0x00	2800
118	0B3A4	<wait>	2800
119	0B3A4	<wait>	D001
120	0B3A4	BEQ 0000B3AA	D001
121	0234D	<wait>	D001
122	0234D	<wait>	D001

To distinguish between ARM and Thumb instructions, a T[15:0], T[31:16], or THUMB notation will appear at the right end of the Invasm field for THUMB instructions. T[15:0] and T[31:16] will appear only with a 32-bit data bus.

Typical AMBH AHB Listing



Display Filtering

The inverse assembler allows you to show or suppress several types of states using a function called *display filtering*. States can be filtered according to their cycle type (e.g. wait or read) or according to which memory region was accessed for the cycle.

NOTE:

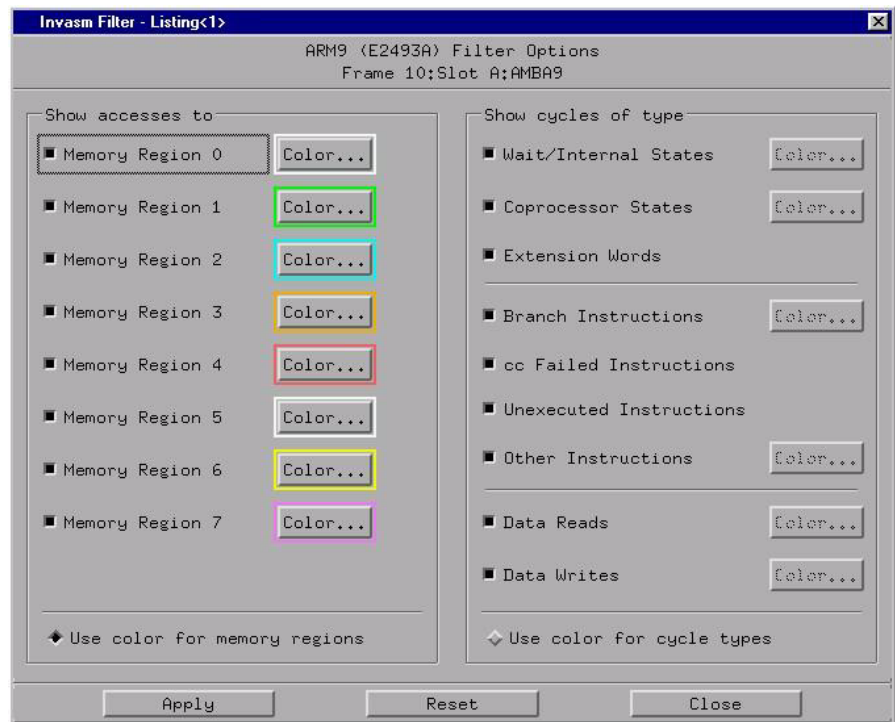
By default, all types of cycles (states) are shown in the listing window. To suppress a cycle (state) of a certain type, un-select its button by clicking it.

The Invasm Filter settings do not affect the data that is stored by the logic analyzer, only whether the data is displayed. For different analysis requirements, the same data can be examined with different settings.

Display filtering allows faster analysis in two ways:

1. Unneeded information can be filtered out of the display. For example, suppressing Wait/Internal states will show only states in which an instruction or data fetch appears.
2. Particular operations can be isolated by suppressing all others. For example, quick analysis of branches can be shown by suppressing all other states.

Display Filtering Dialog—ARM Core and AMBA ASB



“Show access to” section

Memory Region. Select or un-select the check box to the left of **Memory Region 0** through **Memory Region 7** to show or suppress display of each of the memory regions configured in the Memory Map dialog.

Use color for memory regions. Select the **Use color for memory regions** check box to apply color to states in which memory regions are accessed. Use the **Color...** button to choose a different color.

“Show cycles of type” section

This section provides control over which types of cycles are displayed. Select or unselect the associated check box to show or suppress display of cycles.

Wait/Internal States. Wait states can be filtered with or without nWAIT (B_WAIT). The inverse assembler will not mark any states as internal cycles without both nWAIT (B_WAIT) and nMREQ (B_TRAN[1]).

Coprocessor States. In order for these states to be filtered, nMREQ (B_TRAN[1]) and SEQ (B_TRAN[0]) must be connected. A coprocessor state is defined when nMREQ = 1 and SEQ = 1.

Extension words. States that have been used to build up instructions or data fetches in systems with 8 or 16-bit data buses.

Branch Instructions. These can be any instruction that alters the flow of a program. Examples are loads to the PC, conditional branches, moves to the PC, software interrupts, etc.

cc Failed Instructions. These are instructions that have a condition code that did not pass. These instructions are marked with a ‘-’. nEXEC (INSTEEXEC) is required to mark any states as “cc Failed Inst”.

Unexecuted Instructions. These are instructions that were not executed due to a pipeline flush. These instructions are marked with a ‘*’. nOPC (B_PROT[0]) is required to mark any state as “Unexecuted Inst.”.

Other Instructions. This category is for all other instructions that do not fall into any of the above categories.

Data Reads, Data Writes. These states include DMA reads/writes as well as normal core reads/writes.

NOTE:

Memory region filters have precedence over Cycle type filters.

The Agilent 16700-series logic analysis systems provide an additional feature for analyzer data. In addition to showing or suppressing states, the selected states can be shown in color. Color can be used for distinguishing either memory region accesses or cycle types, but not both at the same time.

NOTE:

For proper operation of the software analyzer (SWA), all unexecuted instructions due to pipeline flush should be filtered out.

State Symbols—ARM Core and AMBA ASB

A column to the left of the ARM instruction in the listing window can contain a symbol which provides further explanation of the signal.

State Symbols	
+	Executed instruction due to condition code passed
-	Unexecuted instruction due to condition code failed
*	Unexecuted instruction due to pipeline flush
.	Supervisor mode instruction
!	Memory abort cycle
D	Debug state
@	Unaligned data transfer

Note: a "*" symbol will overwrite a '+' or a '-' symbol

Several of these symbols are shown only if the associated status signal is attached.

*	requires the nOPC signal (B_PROT[0] for AMBA ASB)
+ or -	requires the nEXEC signal (INSTEXEC for ARM9 AMBA ASB)
.	requires the nTRANS signal (B_PROT[1] for AMBA ASB)
!	requires the ABORT signal (B_ERROR for AMBA ASB)
D	requires the DBGACK signal

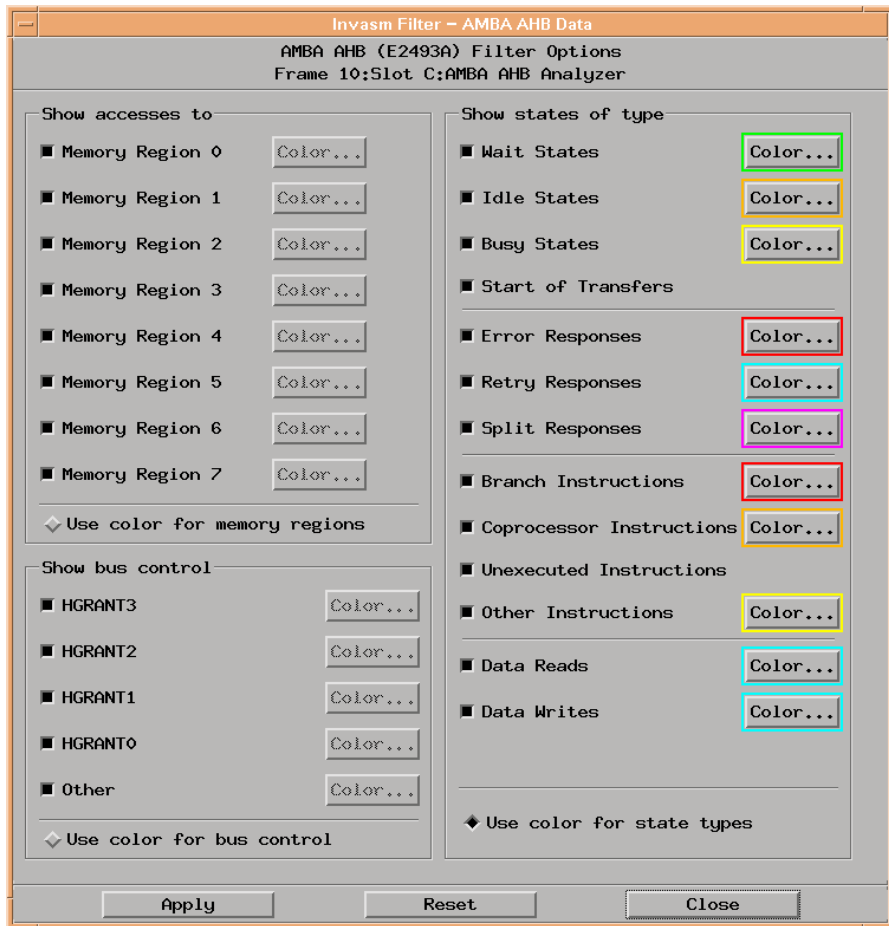
There is an additional symbol for the AMBA ASB inverse assembler only:

R	The current cycle has been completed, but retracted.
---	--

NOTE:

This symbol is only visible with the B_LAST signal.

Display Filtering Dialog—AMBA AHB



“Show access to” section

Memory Region. Select or un-select the check box to the left of Memory Region 0 through Memory Region 7 to show or suppress display of each of the memory regions configured in the Memory Map dialog.

Use color for memory regions. Select the Use color for memory regions check box to apply color to states in which memory regions are accessed. Use the Color... button to choose a different color.

AMBA AHB Show Bus Control

Operation depends upon whether the inverse assembler preference (shown on page 162) is set to Single Master or Multiple Master configuration.

- Single Master Configuration

When the master associated with the HGRANT signal (asserted high) has control of the bus, the **HGRANT0** button is active. States of this type can be shown or hidden by selecting or unselecting the **HGRANT0** button.

When the master associated with the HGRANT signal (low) does not have control of the bus, the **Other** button is active. States of this type can be shown or hidden by selecting or unselecting the **Other** button.

HGRANT [3:1] have no effect.

- Multiple Master Configuration

HGRANT [3:0] are associated with their respective processors. **Other** is associated with states in which processors 3 through 0 do not have control of the bus. Select or unselect the respective check box to show or suppress display of these cycle types.

Use color for bus control. Select the **Use color for bus control** check box to apply color to states in which bus control occurs. Use the **Color...** button to choose a different color.

“Show states of type” section

This section provides control over which types of states are displayed. Select or unselect the associated check box to show or suppress display of states.

Wait States. The slave holds the HREADY line low while it is processing information. The slave asserts HREADY high when the transfer can complete.

Idle States. Indicates that no data transfer is required.

Busy States. Indicate that the next transfer can not take place immediately.

Start of Transfers. Indicates the start of a transfer of a burst or a signal transfer.

Error Responses. An error condition is signaled to the bus master so that it is aware the transfer has been unsuccessful.

Viewing ARM Trace Data

Retry Responses. The RETRY response shows that the transfer has not yet completed. The bus master should retry the transfer.

Split Responses. The transfer has not yet completed successfully. The bus master must retry the transfer when it is next granted access to the bus.

Branch Instructions. These can be any instruction that alters the flow of a program. Examples are loads to the PC, conditional branches, moves to the PC, software interrupts, etc.

Coprocessor Instructions. Includes all instructions issued from the system processor to the coprocessor.

Unexecuted Instructions. These are instructions that were not executed due to a pipeline flush. These instructions are marked with a minus sign (-).

Other Instructions. This category is for all other instructions that do not fall into any of the above categories.

Data Reads, Data Writes. These states include DMA reads/writes as well as normal core reads/writes.

Use color for state types. Select the **Use color for memory regions** check box to apply color to states in which memory regions are accessed. Use the **Color...** button to change the color choice.

NOTE:

Memory region filters have precedence over state type filters.

The Agilent 16700-series logic analysis systems provide an additional feature for analyzer data. In addition to showing or suppressing states, the selected states can be shown in color. Color can be used for distinguishing either memory region accesses or state types, but not both at the same time.

NOTE:

For proper operation of the software analyzer (SWA), all unexecuted instructions due to pipeline flush should be filtered out.

State Symbols—AMBA AHB

A minus sign (-) in the column to the left of the ARM instruction in the listing window indicates an instruction that was not executed due to a pipeline flush.

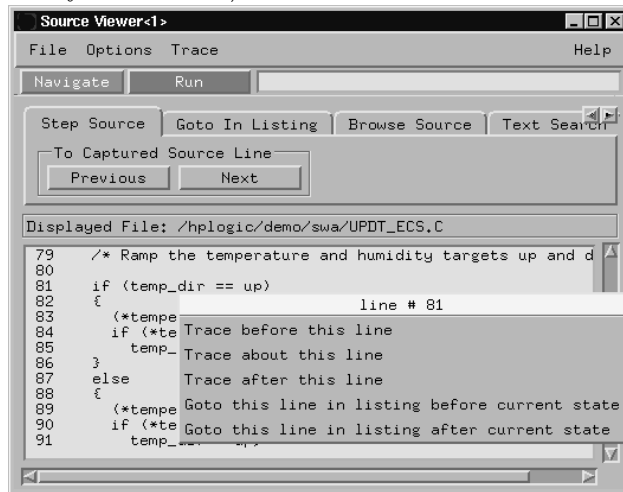
Displaying Source Code

The Agilent B4620B Source Correlation Tool Set lets you:

- View the high-level source code associated with captured data.
- Set up triggers based on source code.

The source correlation tool set correlates the logic analyzer's address label with a line of high-level source code whose address, symbol name, file name, and line numbers are described in a symbol file downloaded to the logic analyzer.

To display the Source Viewer window, select the logic analyzer module icon in the System window, and choose **Source Viewer...**



The first time you display the Source Viewer window, it will probably be blank. To see the source code select the Browse Source tab and choose a file to display. To see source code that corresponds to a particular state in the listing, select that state in the Listing window.

If you purchased a solution, the Agilent B4620B Source Correlation Tool Set was included. Otherwise, the source correlation tool set is available as an add-on product for the Agilent 16700-series logic analysis system and must be licensed before you can use it (see the System Admin dialogs for information on licensing).

See Also

More information on configuring and using the source correlation tool set can be found in the online help for your logic analysis system.

Requirements for source correlation

The source correlation tool set works with many microprocessors and their embedded software development environments.

However, the overall effectiveness of the source correlation tool set will vary to some degree depending on the specific development environment it is being used in. The following areas affect the performance of the source correlation tool set for different development environments:

- Probing connections and inverse assembler.
All the information needed to reconstruct the complete address bus of the target system must be acquired by the logic analyzer.
The logic analyzer's inverse assembler may need to reconstruct any incomplete address bus information and/or filter out any unexecuted instructions.
When displaying the next or previous instances of a source line, the Source Viewer display uses the PC or SW_ADDR (Software Address) label generated by the inverse assembler.
- Object file symbols.
The source correlation tool set requires that symbols be loaded into the logic analyzer (refer to the "Object File Symbols" section earlier in this chapter).
The compiler needs to produce an object file format that is readable by the logic analyzer; otherwise, a general-purpose ASCII (GPA) format file needs to be generated.
- Access to source code files.
The source correlation tool set requires that you give the logic analysis system access to your program's high-level source files (either by NFS mounting the file system that contains the source files or by copying source files to the logic analysis system disk).

Inverse Assembler Generated PC (Software Address) Label

In the Agilent 16700-series logic analysis system, the inverse assembler generates a “SW_ADDR” label. The SW_ADDR label is displayed as another column in the Listing tool. This label is also known as the Software Address generated by the inverse assembler.

The “Goto this line in listing” commands in the Agilent 16700-series logic analysis system perform a pattern search on the SW_ADDR label in the Listing display (when an inverse assembler is loaded). Because the inverse assembler is called for each line that is searched, the search can be slow, especially with a deep memory logic analyzer.

Also, a single source code line will generate many assembly instructions. The “Goto this line in listing” commands will not find a given source code line unless the first assembly instruction generated from the source line has been acquired by the logic analyzer.

For example, if the compiler unrolls a loop in the source code, the trace could begin after the first assembly instruction of the loop has been executed. A “Goto this line in listing” command would not find the source line.

Access to Source Code Files

The source correlation tool set must be able to access the high-level source code files referenced by the symbol information so that these source files can be displayed next to and correlated with the logic analyzer's execution trace acquisition. This requires you to be aware of a number of issues.

Source File Search Path

Verify that the correct file search paths for the source code have been entered into the source correlation tool set. The Agilent B4620B Source Correlation Tool Set can often read and access the correct source code file from information contained in the symbol file, if the source code files have not been moved since they were compiled.

Network Access to Source Files

If source code files are being referenced across a network, the Agilent logic analyzer networking must be compatible with the user's network environment. Agilent logic analyzers currently support Ethernet networks running a TCP/IP protocol and support ftp, telnet, NFS client/server and X-Window client/server applications. Some PC networks may require extensions to the normal LAN protocols to support the TCP/IP protocol and/or these networking applications. Users should contact their LAN system administrators to help setup the logic analyzer on their network.

Source File Version Control

If the source code files are under a source code or version control utility, check the file names and paths carefully. These utilities can change source code file paths and file names. If these names are changed from the information contained in the symbol file, the source correlation tool set will not be able to find the proper source code file. These version control utilities usually provide an "export" command that creates a set of source code files with unmodified names. The source correlation tool set can then be given the correct path to these files.

Viewing ARM core or AMBA ASB Trace Data on the 1660/70-series logic analyzers

This section discusses the general output format of the inverse assembler and processor-specific information.

To display captured state data:

- If you are using an Agilent 16700 logic analysis system, select the analyzer you are using, and then choose Listing... from the pop-up menu.

For all other analyzers, select the listing menu

- Set the base for the DATA label to Invasm.

Here is a typical listing menu:

Acquisition Time
02 Jul 1997 15:34:35

Label>	ADDR	ARM & Thumb Mnemonics v1.00	DATA	STAT	
Base>	Hex	Inverse Assembler	Hex	Hex	
	141	0031FE	BL 00003ABC	THUMB	005E 000040
	142	0031FF	<wait>		00FC 000040
	143	0031FF	<wait>		00FC 000040
	144	0031FF	<extension state>		00FC 000040
	145	003200	<wait>		0080 000040
	146	003200	<wait>		0080 000040
	147	003200	* MOV R0 ,#0x80	THUMB	0080 000040
	148	003201	<wait>		0020 000040
	149	003201	<wait>		0020 000040
	150	003201	<extension state>		0020 000040
	151	003202	<wait>		0003 000040
	152	003202	<wait>		0003 000040
	153	003202	* LDR R1 , [PC, #0x00C]	THUMB	0003 000040
	154	003203	<wait>		0049 000040
	155	003203	<wait>		0049 000040
	156	003203	<extension state>		0049 000040

To distinguish between ARM and Thumb instructions, a T[15:0], T[31:16], or THUMB notation will appear at the right end of the Invasm field for THUMB instructions. T[15:0] and T[31:16] will appear only with a 32-bit data bus.

Coordinating Logic Analysis with
Processor Execution

This chapter describes how to use the emulation and analysis features of your Agilent 16700 logic analysis system to gain insight into your target system.

What are some of the tools I can use?

You can use a combination of all of the following tools to control and measure the behavior of your target system:

- Your logic analyzer, to acquire data from the processor bus while it is running full-speed.
- An emulation module, to control the execution of your target processor and to examine the state of the processor and of the target system.
- The Emulation Control Interface, to control and configure the emulation module, and to display or change target registers and memory.
- Display tools including the Listing tool, Chart tool, and System Performance Analyzer tool to make sense of the data collected using the logic analyzer.
- Your debugger, to control your target system using the emulation module. Do not use the debugger at the same time as the Emulation Control Interface.
- The Agilent B4620B Source Correlation Tool Set, to relate the analysis trace to your high-level source code.

Which assembly-level listing should I use?

Several windows display assembly language instructions. Be careful to use to the correct window for your purposes:

- The Listing tool shows processor states that were captured during a "Run" of the logic analyzer. Those states are disassembled and displayed in the Listing window.
- The Emulation Control Interface shows the disassembled contents of a section of memory in the Memory Disassembly window.
- Your debugger shows your program as it was actually assembled, and (if it supports the emulation module) shows which line of assembly code corresponds to the value of the program counter on your target system.

Which source-level listing should I use?

Different tools display source code for different uses:

- The Source Viewer window allows you to follow how the processor executed code as the analyzer captured a trace. Use the Source Viewer to set analyzer triggers. The Source Viewer window is available only if you have licensed the Agilent B4620B Source Correlation Tool Set.
- Your debugger shows which line of code corresponds to the current value of the program counter on your target system. Use your debugger to set breakpoints.

Where can I find practical examples of measurements?

The Measurement Examples section in the online help contains examples of measurements which will save you time throughout the phases of system development: hardware turn-on, firmware development, software development, and system integration.

A few of the many things you can learn from the measurement examples are:

- How to find glitches.
- How to find NULL pointer de-references.
- How to profile system performance.

To find the measurement examples, select the Help icon in the logic analysis system window, then select "Measurement Examples."

Triggering the Emulation Module from the Analyzer

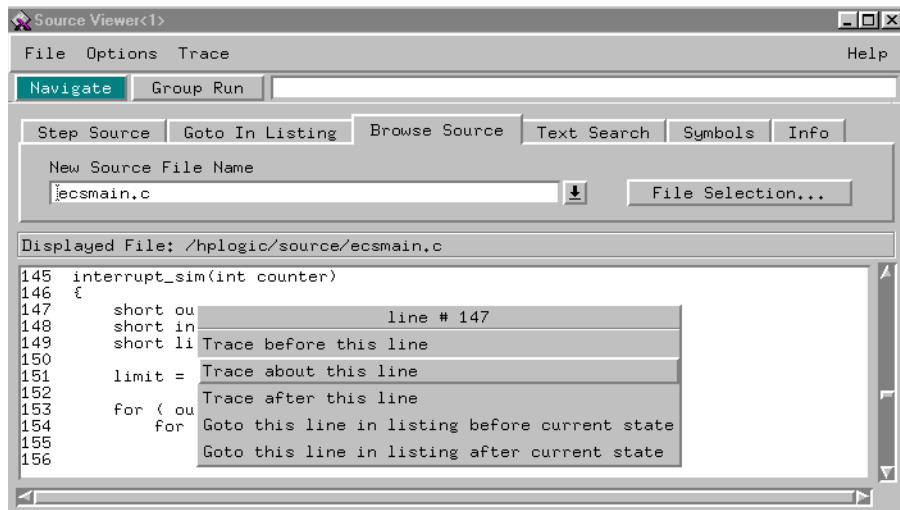
The logic analyzer may be used to signal the emulation module to stop (break) the target processor. This is done from either the Source Viewer window or the Intermodule window. If you are using the Agilent B4620B Source Correlation Tool Set, using the Source Viewer window is the easiest method.

To stop the processor when the logic analyzer triggers on a line of source code (Source Viewer window)

If you have the Agilent B4620B Source Correlation Tool Set, you can easily stop the processor when a particular line of code is reached.

- 1 Click on the logic analyzer module icon in the System window, and choose **Source Viewer...**
- 2 In the Source Viewer window, select the line of source code where you want to set the trigger, then select **Trace about this line**.

The logic analyzer trigger is now set.



3 Select **Trace→Enable - Break Emulator On Trigger**.

The emulation module is now set to halt the processor after receiving a trigger from the logic analyzer.

To disable the processor stop on trigger, select **Trace→Disable - Break Emulator On Trigger**.

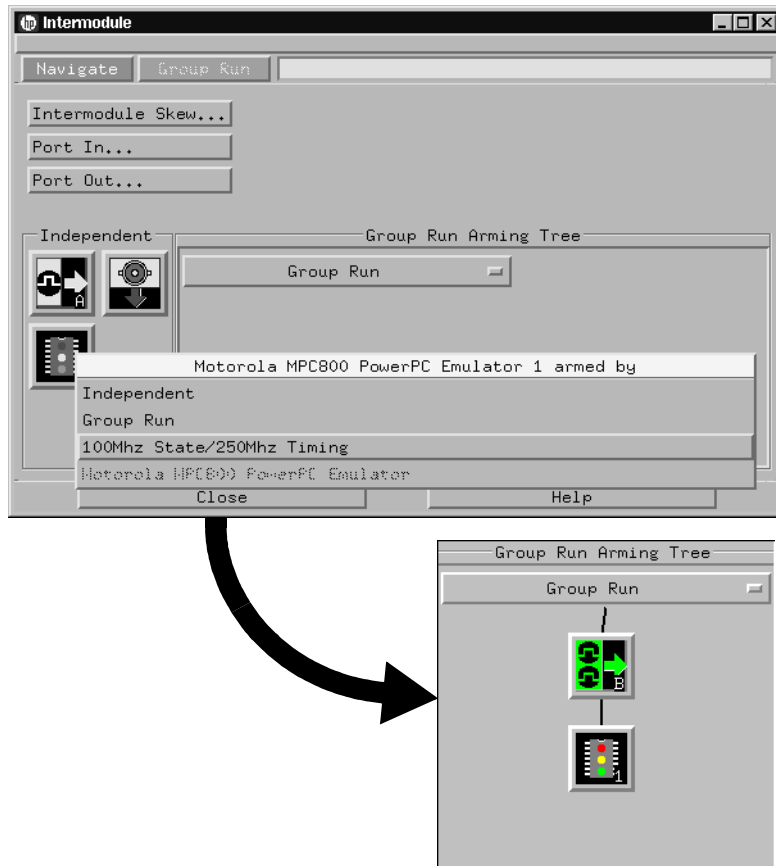
4 Click **Group Run** in the Source window (or other logic analyzer window).

5 If your target system is not already running, select **Run** in the emulation Run Control window to start your target.

To stop the processor when the logic analyzer triggers (Intermodule window)

Use the Intermodule window if you do not have the Agilent B4620B Source Correlation Tool Set or if you need to use a more sophisticated trigger than is possible in the Source Viewer window.

- 1 Create a logic analyzer trigger.
- 2 Click on the **Intermodule** icon in the System window.
- 3 In the Intermodule window, select the emulation module icon, then select the analyzer which is intended to trigger it.



The emulation module is now set to stop the processor when the logic analyzer triggers.

- 4 Click **Group Run** in the Source window (or other logic analyzer window).
- 5 If your target system is not already running, select **Run** in the emulation Run Control window to start your target.

See Also

See the online help for your logic analysis system for more information on setting triggers.

To minimize the "skid" effect

There is a finite amount of time between when the logic analyzer triggers, and when the processor actually stops. During this time, the processor will continue to execute instructions. This latency is referred to as the skid effect.

Skid is greatly increased when the processor is stopped using the JTAG scan chains. Using the DBGRQ and DBGACK signals reduces the "skid" process because the process uses hardware connections instead of software connections.

To minimize the skid effect:

- 1 In the Emulation Control Interface, open the Configuration window.
- 2 Set processor clock speed to the maximum value that your target can support.
- 3 Enable the DBGRQ signal.

The amount of skid will depend on the processor's execution speed and whether code is executing from the cache.

See the *E5900A/B Option 300 Emulation for the ARM7/ARM9 User's Guide* for information on how to configure the clock speed.

To configure the availability of DBGACK and DBGRQ, see page 238.

For more information on the recommended signals, see page 38.

To configure the availability of DBGACK and DBGRQ

The emulation probe will make use of the DBGACK and DBGRQ signals if they are available on the debug port connector.

The DBGACK signal allows the emulator to quickly detect entry or exit from debug mode. Also, the emulator is able to start or stop the logic analyzer through the “TRIGGER OUT” of the emulator.

The DBGRQ signal is used to quickly enter debug mode after receiving a “BREAK IN” signal from the logic analyzer. This allows the logic analyzer triggering capability to be used for complex breakpoints.

To make use of these signals, the emulator must be configured correctly. The following cf commands allow the specification of whether each signal is connected or not.

Processor	Configured for	Built-in command
yes	The corresponding signal is connected and will be used.	cf dbgack=yes cf dbgreq=yes
no	The corresponding signal is not connected and will not be used. (Default)	cf dbgack=no cf dbgreq=no

To stop the analyzer and view a measurement

- To view an analysis measurement you may have to select **Stop** after the trigger occurs.

NOTE:

When the target processor stops it may cause the analyzer qualified clock to stop. Therefore most intermodule measurements will have to be stopped to see the measurement.

Example

An intermodule measurement has been set up where the analyzer is triggering the emulation module. The following sequence could occur:

- 1** The analyzer triggers.
 - 2** The trigger ("Break In") is sent to the emulation module.
 - 3** The emulation module stops the user program which is running on the target processor. The processor enters a background debug monitor.
 - 4** Because the processor has stopped, the analyzer stops receiving a qualified clock signal.
 - 5** If the trigger position is "End", the measurement will be completed.
 - 6** If the trigger position is not "End", the analyzer may continue waiting for more states.
 - 7** The user selects **Stop** in a logic analyzer window, which tells the logic analyzer to stop waiting, and to display the trace.
-

Tracing until the processor halts

If you are using a state analyzer, you can begin a trace, run the processor, then manually end the trace when the processor has halted.

To halt the processor, you can set a breakpoint using the Emulation Control Interface or a debugger.

Some possible uses for this measurement are:

- To store and display processor bus activity leading up to a system crash.
- To capture processor activity before a breakpoint.
- To determine why a function is being called. To do this, you could set a breakpoint at the start of the function then use this measurement to see how the function is getting called.

NOTE:

This kind of measurement is easier than setting up an intermodule measurement trigger.

If you have already set up an intermodule measurement, you must “undo” it by setting all components in the intermodule window to run independently.

To capture a trace before the processor halts

- 1 Set the sampling to **state mode** and the trigger condition to **Run until user stop**.

Now proceed to step 2 under “All Agilent logic analysis systems”.

- 1 In the configuration dialog, set the machine type to **state**, and set the logic analyzer to trigger on **nostate**.

Now proceed to step 2 under “All Agilent logic analysis systems”.

- 2 Set the trigger point (position) to **End**.
- 3 In a logic analyzer window, select **Run**.
- 4 In the Emulation Control Interface or debugger select **Run**.
- 5 When the target processor halts, select **Stop** in the logic analyzer window to complete the measurement.

NOTE:

This is the recommended method to do state analysis of the processor bus when the processor halts.

If you need to capture the interaction of another bus when the processor halts or you need to make a timing or oscilloscope measurement you will need to trigger the logic analyzer from the emulation module (described in the next section).

Triggering the Logic Analyzer from the Emulation Module

You can create an intermodule measurement which will allow the emulation module to trigger another module such as a timing analyzer or oscilloscope.

If you are only using a state analyzer to capture the processor bus then it will be much simpler to use “Tracing until processor halts” as described on page 241.

Before you trigger a logic analyzer (or another module) from the emulation module, you should understand a few things about the emulation module trigger:

The emulation module trigger signal

The trigger signal coming from the emulation module is an "In Background Debug Monitor" (In Monitor) signal. This may cause confusion because a variety of conditions could cause this signal and falsely trigger your analyzer.

The In Monitor trigger signal can be caused by:

- The most common method to generate the signal is to select **Run** and then select **Break** in the Emulation Control Interface. Going from Run (Running User Program) to Break (In Monitor) generates the trigger signal.
- Another method to generate the In Monitor signal is to select **Reset** and then select **Break**. Going from the reset state of the processor to the In Monitor state will generate the signal.
- In addition, an In Monitor signal is generated any time a debugger or other user interface reads a register, reads memory, sets breakpoints or steps. Care must be taken to not falsely trigger the logic analyzers that are listening to the In Monitor signal.

Group Run

The intermodule bus signals can still be active even without a Group Run.

The following setups can operate independently of Group Run:

- Port In connected to an emulation module
- Emulation modules connected in series
- Emulation module connected to Port Out

Here are some examples:

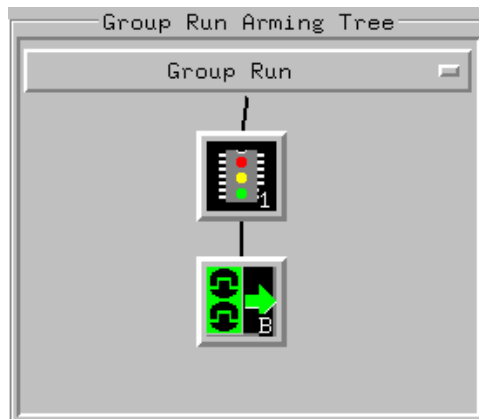
- If Group Run is armed from Port In and an emulation module is connected to Group Run, then any Port In signal will cause the emulation module to go into monitor. The Group Run button does not have to be selected for this to operate.
- If two emulation modules are connected together so that one triggers another, then the first one going into monitor will cause the second one to go into monitor.
- If an emulation module is connected to Port Out, then the state of the emulation module will be sent out the Port Out without regard to Group Run.

The current emulation module state (Running or In Monitor) should be monitored closely when they are part of a Group Run measurement so that valid measurements are obtained.

Group Run into an emulation module does not mean that the Group Run will Run the emulation module.

The emulation module Run, Break, Step, and Reset are independent of the Group Run of the analyzers.

For example, suppose you have the following intermodule measurement set up:



Clicking the **Group Run** button (at the very top of the Intermodule window or a logic analyzer window) will start the analyzer running. The analyzer will then wait for an arm signal. Now when the emulation module transitions into Monitor from Running (or from Reset), it will send the arm signal to the analyzer. If the emulation module is In Monitor when you select **Group Run**, you will then have to go to the emulation module or your debugger interface and manually start it running.

Debuggers can cause triggers

Emulation module user interfaces may introduce additional states into your analysis measurement and in some cases falsely trigger your analysis measurement.

When a debugger causes your target to break into monitor it will typically read memory around the program stack and around the current program counter. This will generate additional states that appear in the listing.

You can often distinguish these additional states because the time tags will be in the μs and ms range. You can use the time tag information to determine when the processor went into monitor. Typically the time between states will be in the nanoseconds while the processor is running and will be in the μs and ms range when the debugger has halted the processor and is reading memory.

Note also that some debugger commands may cause the processor to break temporarily to read registers and memory. These states that the debugger introduces will also show up in the trace listing.

If you define a trigger on some state and the debugger happens to read the same state, then you may falsely trigger your analyzer measurement.

In summary, when you are making an analysis measurement be aware that the debugger could be impacting your measurement.

To trigger the analyzer when the processor halts - timing mode

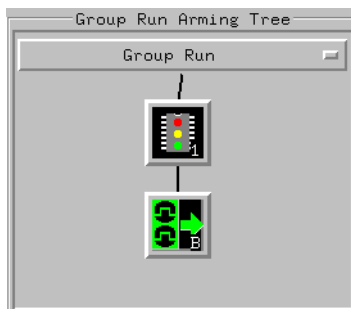
If your processor halts unexpectedly, and you would like to see timing information on your bus prior to the halt, set up this measurement.

The following example shows how to set up an HP/Agilent 16700-series logic analysis system with VisiTrigger. This measurement can also be set up using HP/Agilent 16700-series logic analysis systems without VisiTrigger, and HP/Agilent 1660/1670/16500-series logic analysis systems.

NOTE:

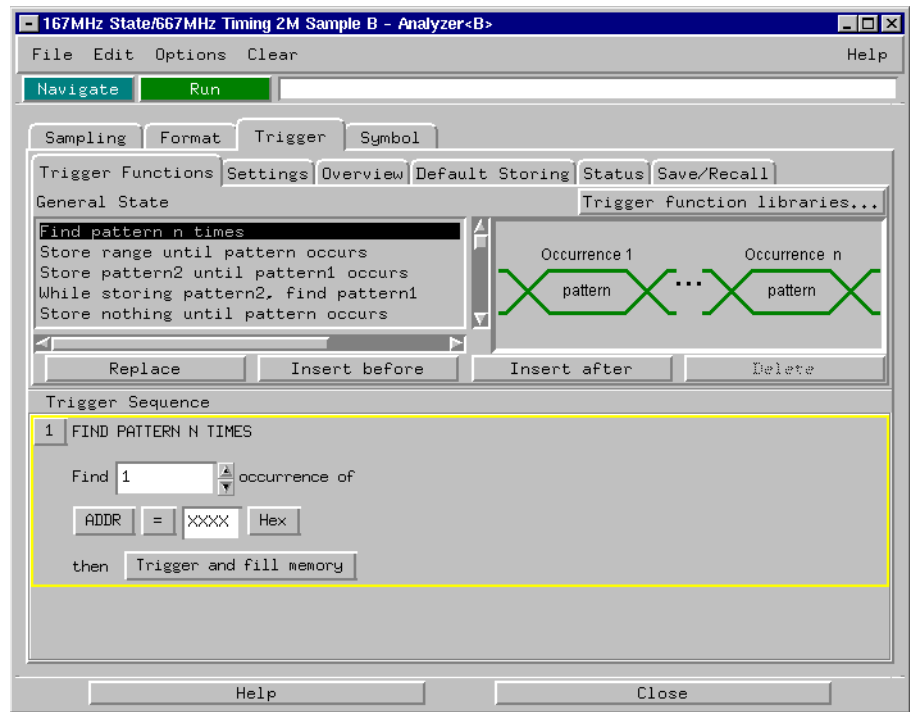
If you only need state information leading up to a processor halt, and timing information is not important, use the procedure called “To capture a trace before the processor halts” on page 241. It is much simpler.

- 1 In the Intermodule window, select the logic analyzer you want to trigger and select the emulation module. A picture (similar to the one shown below) will appear in the intermodule window. This sets the logic analyzer to trigger when the processor halts.



Now continue to step 2.

- 2 Set the sampling mode to timing and set the trigger as shown below:



- 3 Set the trigger position to end.
- 4 Click **Group Run** to start the analyzer(s).
- 5 Click **Run** in the Emulation Control Interface or use your debugger to start the target processor running.

Clicking **Group Run** will *not* start the emulation module. The emulation module run, break, step, reset are independent of the Group Run of the analyzers.

- 6 Wait for the Run Control window in the Emulation Control Interface or the status display in your debugger to show that the processor has halted.

The logic analyzer will store states until the processor halts.

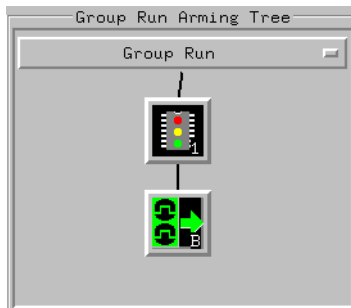
To trigger the analyzer when the processor reaches a breakpoint

This measurement is exactly like the one on the previous page, but with the one additional complexity of setting breakpoints. Be aware that setting breakpoints may cause a false trigger and that the breakpoints set may not be valid after a reset.

NOTE:

If you are only using a state analyzer to capture the processor bus then it will be much simpler to use “Tracing until processor halts” as described on page 241.

- 1 Set the logic analyzer to trigger on **anystate**.
- 2 Set the trigger point to **center** or **end**.
- 3 In the Intermodule window, select the logic analyzer you want to trigger and select the emulation module.



The logic analyzer is now set to trigger on a processor halt.

- 4 Set the breakpoint.

If you are going to run the emulation module from Reset you must do a **Reset** followed by **Break** to properly set the breakpoints. The Reset will clear all on-chip hardware breakpoint registers. The Break command will then reinitialize the breakpoint registers. If you are using software breakpoints that insert an illegal instruction into your program at the breakpoint location you will not need to do the Reset, Break sequence. Instead you must take care to properly insert your software breakpoint in your RAM program location.

- 5 Click **Group Run** to start the analyzer(s).

- 6 Click **Run** in the Emulation Control Interface or use your debugger to start the target processor running.

Clicking **Group Run** will *not* start the emulation module. The emulation module run, break, step, reset are independent of the Group Run of the analyzers.

- 7 Wait for the Run Control window in the Emulation Control Interface or the status display in your debugger to show that the processor has stopped.

The logic analyzer will store states until the processor stops, but may continue running.

You may or may not see a "slow clock" error message. In fact, if you are using a state analyzer on the processor bus the status may never change upon receiving the emulation module trigger (analysis arm). This occurs because the qualified processor clock needed to switch the state analyzer to the next state is stopped. For example, the state analyzer before the arm event may have a status of "**Occurrences Remaining in Level 1: 1**" and after the arm event it may have the same status of "**Occurrences Remaining in Level 1: 1**"

- 8 If necessary, in the logic analyzer window, select **Stop** to complete the measurement.

If you are using a timing analyzer or oscilloscope the measurement should complete automatically when the processor halts. If you are using a state analyzer, select **Stop** if needed to complete the measurement.

Chapter 10: Coordinating Logic Analysis with Processor Execution
Triggering the Logic Analyzer from the Emulation Module

General-Purpose ASCII (GPA) Symbol
File Format

General-Purpose ASCII (GPA) Symbol File Format

General-purpose ASCII (GPA) format files are loaded into a logic analyzer just like other object files, but they are usually created differently.

If your compiler does not include symbol information in the output, or if you want to define a symbol which is not in the object file, you can create an ASCII format symbol file.

Typically, ASCII format symbol files are created using text processing tools to convert compiler or linker map file output that has symbolic information into the proper format.

You can typically get symbol table information from a linker map file to create a General-Purpose ASCII (GPA) symbol file.

Various kinds of symbols are defined in different records in the GPA file. Record headers are enclosed in square brackets; for example, [VARIABLES]. For a summary of GPA file records and associated symbol definition syntax, refer to the “GPA Record Format Summary” that follows.

Each entry in the symbol file must consist of a symbol name followed by an address or address range.

While symbol names can be very long, the logic analyzer only uses the first 16 characters.

The address or address range corresponding to a given symbol appears as a hexadecimal number. The address or address range must immediately follow the symbol name, appear on the same line, and be separated from the symbol name by one or more blank spaces or tabs. Ensure that address ranges are in the following format:

```
beginning address..ending address
```

Example

```
main      00001000..00001009
test      00001010..0000101F
var1      00001E22           #this is a variable
```

This example defines two symbols that correspond to address ranges and one point symbol that corresponds to a single address.

For more detailed descriptions of GPA file records and associated symbol definition syntax, refer to these topics that follow:

- SECTIONS
- FUNCTIONS
- VARIABLES
- SOURCE LINES
- START ADDRESS
- Comments

GPA Record Format Summary

```
[SECTIONS]  
section_name start..end attribute
```

```
[FUNCTIONS]  
func_name start..end
```

```
[VARIABLES]  
var_name start [size]  
var_name start..end
```

```
[SOURCE LINES]  
File: file_name  
line# address
```

```
[START ADDRESS]  
address
```

#Comments

If no record header is specified, [VARIABLES] is assumed. Lines without a preceding header are assumed to be symbol definitions in one of the VARIABLES formats.

Example

This is an example GPA file that contains several different kinds of records:

```
[SECTIONS]
prog      00001000..0000101F
data      40002000..40009FFF
common    FFFF0000..FFFF1000

[FUNCTIONS]
main      00001000..00001009
test      00001010..0000101F

[VARIABLES]
total     40002000  4
value     40008000  4

[SOURCE LINES]
File: main.c
10        00001000
11        00001002
14        0000100A
22        0000101E

File: test.c
5         00001010
7         00001012
11        0000101A
```

SECTIONS

```
[SECTIONS]
section_name start..end attribute
```

Use SECTIONS to define symbols for regions of memory, such as sections, segments, or classes.

`section_name` A symbol representing the name of the section.

`start` The first address of the section, in hexadecimal.

`end` The last address of the section, in hexadecimal.

`attribute` This is optional, and may be one of the following:

- **NORMAL** (default)—The section is a normal, relocatable section, such as code or data.
- **NONRELOC**—The section contains variables or code that cannot be relocated; this is an absolute segment.

Define sections first

To enable section relocation, section definitions must appear before any other definitions in the file.

Example

```
[SECTIONS]
prog          00001000..00001FFF
data          00002000..00003FFF
display_io    00008000..0000801F  NONRELOC
```

If you use section definitions in a GPA symbol file, any subsequent function or variable definitions must be within the address ranges of one of the defined sections. Functions and variables that are not within the range are ignored.

FUNCTIONS

```
[FUNCTIONS]  
func_name start..end
```

Use FUNCTIONS to define symbols for program functions, procedures, or subroutines.

`func_name` A symbol representing the function name.

`start` The first address of the function, in hexadecimal.

`end` The last address of the function, in hexadecimal.

Example

```
[FUNCTIONS]  
main      00001000..00001009  
test      00001010..0000101F
```

VARIABLES

```
[VARIABLES]
var_name  start [size]
var_name  start..end
```

You can specify symbols for variables either by using the address of the variable, the address and the size of the variable, or a range of addresses occupied by the variable. If you specify only the address of a variable, the size is assumed to be one byte.

var_name A symbol representing the variable name.

start The first address of the variable, in hexadecimal.

end The last address of the variable, in hexadecimal.

size This is optional, and indicates the size of the variable, in bytes, in decimal.

Example

```
[VARIABLES]
subtotal  40002000  4
total     40002004  4
data_array 40003000..4000302F
status_char 40002345
```

SOURCE LINES

```
[SOURCE LINES]
File: file_name
line# address
```

Use SOURCE LINES to associate addresses with lines in your source files.

`file_name` The name of a file.

`line#` The number of a line in the file, in decimal.

`address` The address of the source line, in hexadecimal.

Example

```
[SOURCE LINES]
File: main.c
10      00001000
11      00001002
14      0000100A
22      0000101E
```

START ADDRESS

```
[START ADDRESS]  
address
```

address The address of the program entry point, in hexadecimal.

Example

```
[START ADDRESS]  
00001000
```

Comments

```
#comment text
```

Use the # character to include comments in a file. Any text following the # character is ignored. You can put comments on a line alone or on the same line following a symbol entry.

Example

```
#This is a comment.
```

———— Troubleshooting

Chapter 12: Troubleshooting

If you encounter difficulties while making measurements, use this chapter to guide you through some possible solutions. Each heading lists a problem you may encounter, along with some possible solutions.

If you still have difficulty using the analyzer after trying the suggestions in this chapter, please contact your local Agilent Technologies service center.

CAUTION:

Be sure to power down the target system before connecting or disconnecting cables. Otherwise, you may damage circuitry in the analyzer or target system.

Logic Analyzer Problems

This section lists general problems that you might encounter while using the logic analyzer.

Intermittent data errors

This problem is usually caused by poor connections, incorrect signal levels, or marginal timing.

- ❑ Remove and re-seat all cables and probes, ensuring that there are no bent pins or poor probe connections.
- ❑ Adjust the threshold level of the data pod to match the logic levels in the system under test.
- ❑ Use an oscilloscope to check the signal integrity of the data lines.

Clock signals for the state analyzer must meet particular pulse shape and timing requirements. Data inputs for the analyzer must meet pulse shape and setup and hold time requirements.

See Also

See “Capacitive loading” on page 265 for information on other sources of intermittent data errors.

Unwanted triggers

Unwanted triggers can be caused by instructions that were fetched but not executed.

- ❑ Add the prefetch queue or pipeline depth to the trigger address to avoid this problem.

The logic analyzer captures prefetches, even if they are not executed. When you are specifying a trigger condition or a storage qualification that follows an instruction that may cause branching, an unused prefetch may generate an unwanted trigger.

No activity on activity indicators

- ❑ Check for loose cables.
- ❑ Check for poor connections to the target system.

No trace list display

If there is no trace list display, it may be that your trigger specification is not correct for the data you want to capture, or that the trace memory is only partially filled.

- ❑ Check your trigger sequence to ensure that it will capture the events of interest.
- ❑ Try stopping the analyzer; if the trace list is partially filled, this should display the contents of trace memory.

Capacitive loading

Excessive capacitive loading can degrade signals, resulting in incorrect capture by the logic analyzer, or system lockup in the microprocessor.

Careful layout of your target system can minimize loading problems and result in better margins for your design. This is especially important for systems that are running at frequencies greater than 50 MHz.

- ❑ Remove as many pin protectors, extenders, and adapters as possible.
- ❑ If multiple interface solutions are available, use one with lower capacitive loading.

Inverse Assembler Problems

This section lists problems that you might encounter while using the inverse assembler.

When you obtain incorrect inverse assembly results, it may be unclear whether the problem is in the inverse assembler or in your target system. If you follow the suggestions in this section to ensure that you are using the inverse assembler correctly, you can proceed with confidence in debugging your target system.

No inverse assembly or incorrect inverse assembly

This problem may be due to incorrect synchronization, modified configuration, incorrect connections, or a hardware problem in the target system. A locked status line can cause incorrect or incomplete inverse assembly.

- ❑ Ensure that each logic analyzer pod is connected to the correct signals.

There is not always a one-to-one correspondence between analyzer pod numbers and cable numbers. Target systems must supply address (ADDR), data (DATA), and status (STAT) information to the analyzer in a predefined order. Thus, one target system might require that you connect cable 2 to analyzer pod 2, while another will require you to connect cable 5 to analyzer pod 2. See Chapter 1 for connection information.

- ❑ Check the activity indicators for status lines locked in a high or low state.
- ❑ Verify that the STAT, DATA, and ADDR format labels have not been modified from their default values.

These labels must remain as they are configured by the configuration file. Do not change the names of these labels or the bit assignments within the labels.

- ❑ Verify that all microprocessor caches and memory managers have been

disabled.

In most cases, if the microprocessor caches and memory managers remain enabled you should still get inverse assembly. It may be incorrect because a portion of the execution trace was not visible to the logic analyzer.

- ❑ Verify that storage qualification has not excluded storage of all the needed opcodes and operands.

Inverse assembler will not load or run

You need to ensure that you have the correct system software loaded on your analyzer.

- ❑ Ensure that the inverse assembler is on the same disk as the configuration files you are loading.

Configuration files for the state analyzer contain a pointer to the name of the corresponding inverse assembler. If you delete the inverse assembler or rename it, the configuration process will fail to load the inverse assembler.

Inverse Assembler Error Messages

“IA Error - Address not in memory map”

This error means that the current address does not fall within any of the memory ranges specified in the memory map. In order for the inverse assembler to work properly, the current address must fall within one of the regions in the memory map. A good way to prevent this message from appearing is to create a “default” memory range. The inverse assembler starts comparing at region 0 and continues downward until a region matches. By placing a base address of 0 and an end address of FFFFFFFF in the last region (region 7), a “catch-all” region is created and any address that does not match with any of the previous 7 regions will match the last one.

“IA Error - Search limited by depth”

This error should only occur when certain status signals are missing. The error occurs because the inverse assembler is either looking forward or backward in the trace data to try to reconstruct the status signals. However, if there isn’t enough trace data to determine what a signal should be, then the error is emitted.

“Inverse Assembler Not Found”

This error occurs if you rename or delete the inverse assembler file that is attached to the configuration file.

- ❑ Ensure that the inverse assembler file is not renamed or deleted, and that it is located in the same directory as the configuration file.

“No Configuration File Loaded”

This is usually caused by trying to load a configuration file for one type of module/system into a different type of module/system.

- ❑ Verify that the appropriate module has been selected from the **Load {module} from File {file name}** in the Agilent Technologies 16500A/B disk operation menu. Selecting **Load {All}** will cause incorrect operation when loading most configuration files.
-

“Selected File is Incompatible”

This occurs when you try to load a configuration file for the wrong module.

- ❑ Ensure you are loading the appropriate configuration file for your analyzer.
 - ❑ Ensure that you have selected the correct destination machine.
-

“Slow or Missing Clock”

- ❑ This error message might occur if the logic analyzer cards are not firmly seated in the Agilent Technologies frame. Ensure that the cards are firmly seated.
- ❑ This error might occur if the target system is not running properly. Ensure that the target system is on and operating properly.
- ❑ Ensure there is activity on the selected clock.
- ❑ If the error message persists, check that the logic analyzer pods are connected to the proper connectors.

“Waiting for Trigger”

If a trigger pattern is specified, this message indicates that the specified trigger pattern has not occurred. Verify that the triggering pattern is correctly set.

- ❑ Setting a trigger can be very complex.
- ❑ Consider how many address lines from the ARM core are pinned out.
- ❑ The processor instruction mode (ARM or THUMB) affects word width.
- ❑ You may need to set one or more least significant trigger address bits to “don’t care” because all instruction fetches are word (ARM) or half-word (THUMB) aligned, and the processor core ignores the two least significant bits in ARM mode and the least significant bit in THUMB mode.
- ❑ The data bus width of the target system may be different from the width of the data operation. You need to know where the data will appear across the width of the data bus. For example: the data operation could be 8-bit and the target’s bus could be 32-bit, so the data could appear in one of four places on the data bus.

Understanding the Impact of ARM Signal Availability

The inverse assembler does the best job it can with limited status signals available. However, in a few cases, the inverse assembler will display states incorrectly if a necessary status signal is missing. The following are some examples of the problems.

LDR PC and LDM PC instructions without nMREQ

After a LDR or LDM instruction where the PC is loaded (this is used for a branch or subroutine return), the ARM core adds an internal cycle. The address driven on the address bus during this cycle is the address of the next instruction that would be fetched if the branch did not take place. Without nMREQ, the inverse assembler cannot detect internal cycles, but in all other cases will decode internal cycles as wait states. In the LDR PC and LDM PC case, the inverse assembler will decode the internal cycle as an opcode fetch or data access (depending on signals available and the state of the core). In the following example, the internal cycle is displayed as a READ WORD in state 34.

100/500MHz LA C
Listing 1
Invasm Options
Print
Run

Markers Off
Acquisition Time
01 Aug 1997 14:56:26

Label>	ADDR	ARM & Thumb Mnemonics v1.00	DATA
Base>	Hex	Inverse Assembler	Hex
20	000080C0	LDR PC, [R13]	E59DF000
21	000080CC	<wait>	E59DF000
22	000080CC	<wait>	E59DF000
23	000080CC	READ WORD 0x000080A0	000080A0
24	000080C4	<wait>	000080A0
25	000080C4	<wait>	000080A0
26	000080C4	* ANDEQ R0, R2, R6, LSR#0x20	00020026
27	00000400	<wait>	00008088
28	00000400	WRITE WORD 0x00008088	00008088
29	000080C8	<wait>	00008088
30	000080C8	<wait>	00008088
31	000080C8	* LDRPLB R5, [R5, #-0x555]	55555555
32	00000400	<wait>	55555555
33	00000400	READ WORD 0x00008088	00008088
34	000080CC	READ WORD 0x00008088	00008088
35	00008088	<wait>	00008088

Problems with literal pools and no nOPC signal

The ARM processor can only provide a limited set of immediate values directly. To load other values, these values are stored in “literal pools” and are loaded indirectly. The literal pools are usually placed in between functions in regions that are otherwise only code. Without the nOPC signal, the inverse assembler will decode these literal pool data reads as instructions. These incorrectly decoded instructions also confuse the algorithms that determine unexecuted prefetches. For this reason, marking of unexecuted prefetches is disabled without the nOPC signal. The SWA tool will indicate inaccurate sequences of source references since it uses all instructions marked as executed, which will include the literal pool data accesses and unexecuted prefetches.

These extra “instructions” that are really data accesses can cause the algorithms used by the inverse assembler to determine size and read/write of data accesses to pick one of these incorrect instructions for a data access and display the size and read/write status incorrectly.

To determine what is really going on, a memory map entry can be added to look at the trace display around a particular literal pool (there are not enough memory map entries to handle all literal pools and the pools will move as code is modified and recompiled). A recommendation is to reserve the first memory map entry for literal pools since it has higher priority than later entries in the memory map, and will overlay ranges specified in later entries so that they will not have to be modified to work correctly.

SWP instructions without nRW, nWAIT and nMREQ

In order to properly detect the read access of a SWP instruction, the inverse assembler requires nRW, nWAIT and nMREQ. If one or more of these signals is missing, the read data access of the SWP instruction will be incorrectly decoded as a wait state as in the following display, state 340 should be displayed as READ WORD].

100/500MHz LA C		Listing 1	Invasm Options	Print	Run
Markers Off		Acquisition Time 06 Aug 1997 09:57:02			
Label>	ADDR	nRW	ARM & Thumb Mnemonics v1.00		DATA
Base>	Hex	Hex	Inverse Assembler		Hex
333	000081C0	0	+ SWP	R1,R2,[R0]	E100
334	000081C4	0	<wait>		E100
335	000081C4	0	+ MOV	RO,RO	E1A0
336	000081C8	0	<wait>		E1A0
337	000081C8	0	+ MOV	RO,RO	E1A0
338	00008214	0	<wait>		E1A0
339	00008214	0	<wait>		E1A0
340	00008214	0	<wait>		0000
341	00008214	1	<wait>		0000
342	00008214	1	<wait>		0000
343	00008214	1	WRITE WORD	0x00000002	0000
344	000081CC	0	<wait>		0000
345	000081CC	0	<wait>		0000
346	000081CC	0	+ MOV	RO,RO	E1A0
347	000081D0	0	<wait>		E1A0
348	000081D0	0	+ STMDB	R13!, CR1-R6,R14}	E92D

Missing MAS[1] and incorrect memory map entries

The inverse assembler will use the memory map entries to determine if an instruction is ARM or THUMB if the MAS[1] signal is not connected. The regions of ARM vs. THUMB code can move as new code is being written. If the memory map entries are not kept up to date, the inverse assembler will incorrectly decode ARM vs. THUMB instructions.

DMA accesses showing up as OPCODES or DATA

If the target system uses DMA and does not provide the DMA signal, the logic analyzer will incorrectly display DMA cycles as code or data. This will cause many problems for the algorithms used by the inverse assembler. The DMA signal is required for systems using DMA.

Store qualification not storing states needed by the inverse assembler for proper operation

For systems with less than 32-bit data buses, the memory controller places the ARM core in a wait state until the complete data bus is fetched from the target system. If wait states are not stored by the logic analyzer, only the last cycle of each memory operation will be stored. This is not enough information for the inverse assembler to operate correctly. Any inverse assembly produced will not be correct. Do not set up the logic analyzer to store only non-wait states with systems that use 8 or 16 bit data buses.

The inverse assembler looks backward in the trace buffer to determine nRW and MAS signals for data accesses if these signals are not connected. If the instructions which caused the data access or any instructions between these two states are not stored, the inverse assembler may decode these status lines incorrectly.

Other problem scenarios

The inverse assembler trace looks like there are several instructions being decoded for the same address

This may be caused by the inversion of the wait bit. Make sure that the menu option in the signals menu is correct. For active low wait signals, the inversion choice should be 'No'. For active high wait signals, the inversion choice should be 'Yes'.

Acquisition Time
01 Aug 1997 08:27:29

Label>	ADDR	nOPC	ARM & Thumb Mnemonics v1.00		DATA
Base>	Hex	Hex	Inverse Assembler		Hex
	1487	008EA0	0	* ANDEQ R8,R0,R0,LSR#0x1D	00008E
	1488	008EA0	0	* ANDEQ R8,R0,R0,LSR#0x1D	00008E
	1489	008EA0	0	* ANDEQ R8,R0,R0,LSR#0x1D	00008E
	1490	008EA0	0	+ BLMI 00009210	4B0000
	1491	008EA4	0	* BLMI 00009214	4B0000
	1492	008EA4	0	* BLMI 00009214	4B0000
	1493	008EA4	0	* BLMI 00009214	4B0000
	1494	008EA4	0	- MOV R4,R0	E1A040
	1495	008EA8	0	* MOV R4,R0	E1A040
	1496	008EA8	0	* MOV R4,R0	E1A040
	1497	008EA8	0	* MOV R4,R0	E1A040
	1498	008EA8	0	+ LDR R0,000081C0	E51F0C
	1499	008EAC	0	* LDR R0,000081C4	E51F0C
	1500	008EAC	0	* LDR R0,000081C4	E51F0C
	1501	008EAC	0	* LDR R0,000081C4	E51F0C
	1502	008EAC	0	+ LDR R0,[R0]	E59000

Some data fetches in the trace listing seem to be shown as instructions

This is the most common problem that occurs when the nOPC signal is not connected. When a program is being compiled the compiler attempts to internally create the immediate data for instructions that require it. However, there are cases when the compiler can't create the required immediate data values.

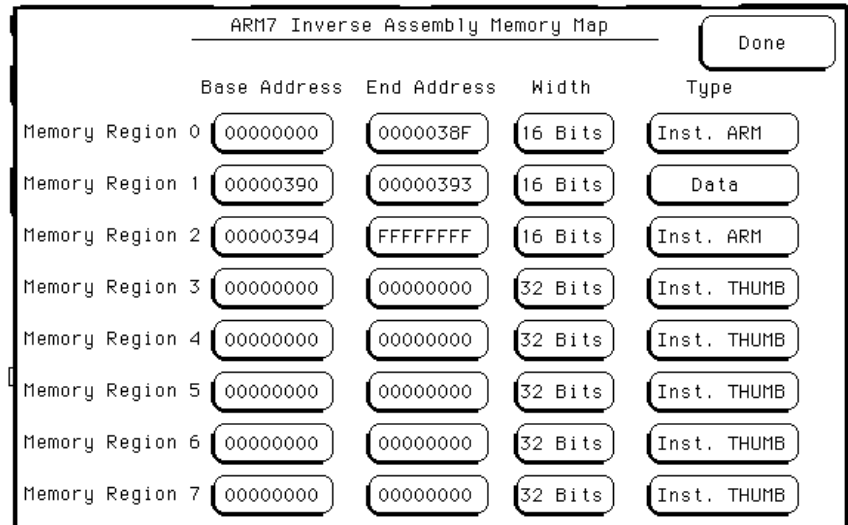
As a result, the compiler places a literal pool close to the instruction that requires the immediate data. This literal pool contains the immediate value that is needed by the instruction. This has the unpleasant effect of having segmented code and data regions that are difficult to find by the user. An example of this behavior is shown below. In order to determine where literal pools exist, use the ARM Project Manager to view the object files created by the compiler

```

C:\devel\arm210\examples\Debug\temp.o (dry_api.Debug)
0x000364: e59f0024 $... : LDR    r0,0x390
0x000368: ebffff24 $... : BL     Proc_8
0x00036c: e51f0280 .... : LDR    r0,0xf4
0x000370: e5900000 .... : LDR    r0,[r0,#0]
0x000374: ebffff21 !... : BL     Proc_1
0x000378: e3a08041 A... : MOV    r8,#0x41
0x00037c: e51f0290 .... : LDR    r0,0xf4
0x000380: e5d00014 .... : LDRB   r0,[r0,#0x14]
0x000384: e1580000 ..X. : CMP    r8,r0
0x000388: da000004 .... : BLE    0x3a0
0x00038c: ea00001e .... : B      0x40c
x$litpool$3
x$litpool_e$3-0x3
0x000390: 00000000 .... : ANDEQ  r0,r0,r0
0x000394: e2880001 .... : ADD    r0,r8,#1
0x000398: e20080ff .... : AND    r8,r0,#0xff
0x00039c: eaffffff6 .... : B      0x37c
0x0003a0: e1a00008 .... : MOV    r0,r8
0x0003a4: e3a01043 C... : MOV    r1,#0x43
0x0003a8: ebffff14 .... : BL     Func_1
0x0003ac: e5dd1040 @... : LDRB   r1,[r13,#0x40]
0x0003b0: e1300001 ..0. : TEQ    r0,r1
0x0003b4: 1a00000b .... : BNE    0x3e8
0x0003b8: e28d1040 @... : ADD    r1,r13,#0x40
0x0003bc: e3a00000 .... : MOV    r0,#0

```

The memory map menu setup for this segment of code would be:



A data state is immediately repeated by another data state or an instruction fetch

If nMREQ is unavailable, internal cycles may be masked over by data states or instruction states.

State symbols (*) for the unused prefetch states don't seem be correct all of the time

In order to have 100% correct marking of all unused prefetch states, nOPC, nRW, nMREQ, nWAIT, and SEQ must be connected. Otherwise, using heuristics, the IA attempts to determine where a branch has occurred. This method may not always work. In cases when there is a branch over two instructions, the IA will not mark the unused instructions correctly. Any marking of unexecuted prefetches requires the nOPC signal.

An extra extension state for an instruction fetch is seen when using reverse memory controller

The memory controller will output an extra decode cycle when a non-sequential access has occurred. This extra extension state is simply the extra decode cycle.

Intermodule Measurement Problems

Some problems occur only when you are trying to make a measurement involving multiple modules.

An event wasn't captured by one of the modules

If you are trying to capture an event that occurs very shortly after the event that arms one of the measurement modules, it may be missed due to internal analyzer delays. For example, suppose you set an oscilloscope module to trigger upon receiving a trigger signal from the logic analyzer because you are trying to capture a pulse that occurs right after the analyzer's trigger state. If the pulse occurs too soon after the analyzer's trigger state, the oscilloscope will miss the pulse.

- ❑ Adjust the skew in the Intermodule menu.

You may be able to specify a skew value that enables the event to be captured.

- ❑ Change the trigger specification for modules upstream of the one with the problem.

If you are using a logic analyzer to trigger an oscilloscope module, try specifying a trigger state one state before the one you are using. This may be more difficult than working with the skew because the prior state may occur more often and not always be related to the event you are trying to capture with the oscilloscope.

“No Configuration File Loaded”

This is usually caused by trying to load a configuration file for one type of module/system into a different type of module/system.

- ❑ Verify that the appropriate module has been selected when you load the configuration file. Selecting Load {All} will cause incorrect operation when loading most configuration files.

See Also

See “To load configuration files (and the inverse assembler) from hard disk—16700-series logic analysis systems” on page 143 or “To load configuration and inverse assembler files—Agilent 16700 logic analysis systems” on page 182.

“Selected File is Incompatible”

This occurs when you try to load a configuration file for the wrong module.

- ❑ Ensure that you are loading the appropriate configuration file for your logic analyzer.
-

“Slow or Missing Clock”

- ❑ This error message might occur if the logic analyzer cards are not firmly seated in the logic analysis system frame. Ensure that the cards are firmly seated.
- ❑ This error might occur if the target system is not running properly. Ensure that the target system is on and operating properly.
- ❑ If the error message persists, check that the logic analyzer pods are connected to the proper connectors on the target system.

NOTE:

See Chapter 5, “Probing the Target System,” beginning on page 121, to determine the proper connections.

“Waiting for Trigger”

If a trigger pattern is specified, this message indicates that the specified trigger pattern has not occurred. Verify that the triggering pattern is correctly set.

- ❑ Setting a trigger can be very complex.
- ❑ Consider how many address lines from the ARM core are pinned out.
- ❑ The processor instruction mode (ARM or THUMB) affects word width.
- ❑ You may need to set one or more least significant trigger address bits to “don’t care” because all instruction fetches are word (ARM) or half-word (THUMB) aligned, and the processor core ignores the two least significant bits in ARM mode and the least significant bit in THUMB mode.
- ❑ The data bus width of the target system may be different from the width of the data operation. You need to know where the data will appear across the width of the data bus. For example: the data operation could be 8-bit and the target’s bus could be 32-bit, so the data could appear in one of four places on the data bus.

Chapter 12: Troubleshooting
Intermodule Measurement Problems

Specifications and Characteristics

This chapter contains specifications and characteristics for the inverse assembler.

The following operating characteristics are not specifications, but are typical operating characteristics for the Agilent Technologies E2493A ARM inverse assembler.

Operating Characteristics — Inverse Assembler

Microprocessor/bus Compatibility	For inverse Assembly: ARM6, ARM7XX, and ARM9 (ARM9 in AMBA mode only).
Clock Speed	Maximum clock speed depends upon the analyzer module installed in the logic analysis system. Here are two examples of maximum clock speeds: 16557D - Maximum clock speed is 140 MHz for 1-4 modules; 100 MHz for 5 modules. 16719A - Maximum clock speed is 333 MHz, with some trade-offs in features. See specification sheet for details.
Probes Required	8-bit bus systems require four 16-channel probes 16/32-bit bus systems require six 16-channel probes
Signal Line Loading	Typically 100 k ohm, plus 10 pF
Setup/Hold Requirement	Data must be valid for a 3.5 ns window with respect to the logic analyzer clock.

Glossary

Analysis Probe A probing solution connected to the target microprocessor. It provides an interface between the signals of the target microprocessor and the inputs of the logic analyzer. Formerly called a "preprocessor."

Background Debug Monitor Also called Debug Mode, In Background, and In Monitor. The normal processor execution is suspended and the processor waits for commands from the debug port. The debug port commands include the ability to read and write memory, read and write registers, set breakpoints and start the processor running (exit the Background Debug Monitor).

Debug Mode See *Background Debug Monitor*.

Debug Port A hardware interface designed into a microprocessor that allows developers to control microprocessor execution, set breakpoints, and access microprocessor registers or target system memory using a tool like the emulation probe.

Elastomeric Probe Adapter A connector that is fastened on top of a target microprocessor using a retainer and knurled nut. The conductive elastomer on the bottom

of the probe adapter makes contact with pins of the target microprocessor and delivers their signals to connection points on top of the probe adapter.

Emulation Migration The hardware and software required to use an emulation probe with a new processor family.

Emulation Module An emulation module is installed within the mainframe of a logic analysis system. An E5901A emulation module is used with a *target interface module* (TIM) or an analysis probe. An E5901B emulation module is used with an E5900B *emulation probe* and does not use a TIM.

Emulation Probe An emulation probe is a standalone instrument connected via LAN to the mainframe of a logic analyzer or to a host computer. It provides run control within an emulation and analysis test setup. Formerly called a "processor probe" or "software probe."

Emulator An emulation module or an emulation probe.

Extender A part whose only function is to provide connections from one location to another. One or more extenders might be stacked to

Glossary

raise a probe above a target microprocessor to avoid mechanical contact with other components installed close to the target microprocessor. Sometimes called a "connector board."

Flexible Adapter Two connection devices coupled with a flexible cable. Used for connecting probing hardware on the target microprocessor to the analysis probe.

Gateway Address An IP address entered in integer dot notation. The default gateway address is 0.0.0.0, which allows all connections on the local network or subnet. If connections are to be made across networks or subnets, this address must be set to the address of the gateway machine.

General-Purpose Flexible Adapter A cable assembly that connects the signals from an elastomeric probe adapter to an analysis probe. Normally, a male-to-male header or transition board makes the connections from the general-purpose flexible adapter to the analysis probe.

High-Density Adapter Cable A cable assembly that delivers signals from an analysis probe hardware interface to the logic analyzer pod

cables. A high-density adapter cable has a single *MICTOR connector* that is installed into the analysis probe, and two cables that are connected to corresponding odd and even logic analyzer pod cables.

High-Density Termination Adapter Cable Same as a High-Density Adapter Cable, except it has a termination in the *MICTOR connector*.

In Background, In Monitor See *Background Debug Monitor*.

Inverse Assembler Software that displays captured bus activity as assembly language mnemonics. In addition, inverse assemblers may show execution history or decode control busses.

IP address Also called Internet Protocol address or Internet address. A 32-bit network address. It is usually represented as decimal numbers separated by periods; for example, 192.35.12.6.

Jumper Moveable direct electrical connection between two points.

JTAG (OnCE) port See *debug port*.

Label Labels are used to group and

Glossary

identify logic analyzer channels. A label consists of a name and an associated bit or group of bits.

Link-Level Address The unique address of the LAN interface. This value is set at the factory and cannot be changed. The link-level address of a particular piece of equipment is often printed on a label above the LAN connector. An example of a link-level address in hexadecimal: 0800090012AB. Also known as an LLA, Ethernet address, hardware address, physical address, or MAC address.

Mainframe Logic Analyzer A logic analyzer that resides on one or more board assemblies installed in a 16500, 1660-series, or 16600/700-series mainframe.

Male-to-male Header A board assembly that makes point-to-point connections between the female pins of a flexible adapter or transition board and the female pins of an analysis probe.

MICTOR Connector A high-density matched impedance connector manufactured by AMP Corporation. *High-density adapter cables* can be used to connect the logic analyzer to MICTOR connectors on the target system.

Monitor, In See *Background Debug Monitor*.

Pod A collection of logic analyzer channels associated with a single cable and connector.

Preprocessor See *Analysis Probe*.

Preprocessor Interface See *Analysis Probe*.

Probe Adapter See *Elastomeric Probe Adapter*.

Processor Probe See *Emulation Probe*.

Run Control Probe See *Emulation Probe* and *Emulation Module*.

Setup Assistant Wizard software program which guides a user through the process of connecting and configuring a logic analyzer to make measurements on a specific microprocessor. The setup assistant icon is located in the main system window.

Shunt Connector. See *Jumper*.

Solution A set of tools for debugging your target system. A solution includes probing, inverse assembly, the B4620B Source Correlation Tool

Glossary

Set, and an emulation module.

Stand-Alone Logic Analyzer A standalone logic analyzer has a predefined set of hardware components which provide a specific set of capabilities. A standalone logic analyzer differs from a mainframe logic analyzer in that it does not offer card slots for installation of additional capabilities, and its specifications are not modified based upon selection from a set of optional hardware boards that may be installed within its frame.

State Analysis A mode of logic analysis in which the logic analyzer is configured to capture data synchronously with a clock signal in the target system.

Subnet Mask A subnet mask blocks out part of an IP address so the networking software can determine whether the destination host is on a local or remote network. It is usually represented as decimal numbers separated by periods; for example, 255.255.255.0.

Symbol Symbols represent patterns and ranges of values found on labeled sets of bits. Two kinds of symbols are available:

1) Object file symbols — Symbols from your source code, and symbols

generated by your compiler. Object file symbols may represent global variables, functions, labels, and source line numbers.

2) User-defined symbols — Symbols you create.

Target Board Adapter A daughter board inside the E5900B emulation probe which customizes the emulation probe for a particular microprocessor family. The target board adapter provides an interface to the ribbon cable which connects to the debug port on the target system.

Target Control Port An 8-bit, TTL port on a logic analysis system that you can use to send signals to your target system. It does not function like a pattern generator or emulation module, but more like a remote control for the target's switches.

Target Interface Module A small circuit board which connects the 50-pin cable from an E5901A emulation module or E5900A emulation probe to signals from the debug port on a target system. Not used with the E5900B emulation probe.

TIM See *Target Interface Module*.

Timing Analysis A mode of logic analysis in which the logic analyzer is configured to capture data at a rate

Glossary

determined by an internal sample rate clock, asynchronous to signals in the target system.

Transition Board A board assembly that obtains signals connected to one side and rearranges them in a different order for delivery at the other side of the board.

Trigger Specification A set of conditions that must be true before the instrument triggers. See the printed or online documentation of your logic analyzer for details.

1/4-Flexible Adapter An adapter that obtains one-quarter of the signals from an elastomeric probe adapter (one side of a target microprocessor) and makes them available for probing.

A

ABORT signal, 160
addresses
 offset, 194
 PC label, 227
Agilent Technologies B4620B
 source correlation tool set, 3
Agilent Technologies E9495A
 emulation solution, 2
analysis probe
 definition, 285
analyzer modes, 153, 171
analyzer problems, 263
 capacitive loading, 265
 intermittent data errors, 263
 unwanted triggers, 264
ASCII format (GPA), 252
assistant
 See setup assistant

B

B4620B source correlation tool set,
 3
background debug monitor, 285
branches, displaying, 218
breakpoints
 tracing until, 248
bus control, AHB, 223

C

CD-ROM, installing software from,
 119
characteristics, 283
 inverse assembler, 284
checklist
 setup, 21
chip selects, 160
clocks
 qualified, and emulator, 239
 slow, 247, 249, 280
colors, 218
comments, in GPA files, 260

configuration
 checklist, 21
configuration files
 loading, 183
configuring menu-logic analyzer,
 155, 172
configuring the logic analyzer, 141,
 142, 169, 181
connecting-to logic analyzer, 122
connecting-to target, 34, 86
connection
 emulation module, 117
 setup checklist, 21
connections, logic analyzer, 124
connector
 JTAG, designing, 84, 112
connector board, 285
connector, MICTOR, 37, 87
connector-choosing a, 35, 86
connector-high density, 37, 87
connector-medium density, 40, 90
creating GPA symbol files, 252
custom probing
 designing connectors, 121

D

debug mode, 285
debug mode-example, 203, 212
debug port, 285
 definition, 285
directories
 configuration files, 143, 182
 source code, 228
display
 bus control, 223
display filtering, 218
DMA signal, 160

E

elastomeric probe adapter
 definition, 285
Emulation Control Interface

 when to use, 232
emulation migration
 definition, 285
emulation module
 definition, 285
 product numbers, 4
emulation probe
 definition, 285
emulation solution, 2
 See solution
endian mode, 160
enhanced inverse assembler
 logic analyzer requirements, 31
equipment required, 25
equipment supplied
 emulation module, 32
 ordering information, 4
 overview, 4
examples, measurement, 233
extender, 285

F

files
 loading vs. installing, 118
filter
 bus control, 223
filtering, display, 218
flexible adapter
 definition, 286
floppy disks
 duplicating, 183
flowchart
 setup, 21
full solution, 2
FUNCTIONS in GPA format, 257

G

gateway address
 definition, 286
General-Purpose ASCII format, 252
 address format, 252
 comments, 260

- F**
FUNCTIONS, 257
record format summary, 254
record headers, 252
SECTIONS, 256
simple form, 252
SOURCE LINES, 259
START ADDRESS, 260
VARIABLES, 258
general-purpose flexible adapter
definition, 286
- H**
high-density adapter cable
definition, 286
high-density termination adapter
definition, 286
- I**
information sources, 6
installation, software, 113
intermodule measurement
creating, 236
intermodule measurement
problems, 279
an event wasn't captured, 279
analyzer doesn't stop, 239
Invasm menu, 154
inverse assembler
definition, 286
loading, 154
loading files, 143, 144, 182, 183
microprocessors supported, 284
operating characteristics, 284
probing for "IA-only", 121
requirements for enhanced, 31
unloading, 154
inverse assembly
displays, 233
IP address
definition, 286
- J**
JTAG
designing connector, 84, 112
jumper, definition, 286
- L**
labels
definition, 286
link-level address
definition, 287
listing window, 216
listing windows, 233
Load menu, 154
loading configurations, vs.
installing, 118, 143
logic analyzer
trigger setup, 191
logic analyzers
16600A and 16700-series, 22
configuring, 143, 144, 182, 183
software version requirements,
31
logic analyzers-compatible, 25
- M**
mainframe logic analyzer
definition, 287
male-to-male header
definition, 287
measurement examples, 233
measurements-making common,
197
memory map, 156
microprocessors supported, 4
MICTOR, 37, 87
MICTOR connector, definition, 287
minus sign in listing, 224
monitor, background debug, 285
- O**
object module file symbols, 165
offset, address, 194
online configuration help, 22
operating characteristics
inverse assembler, 284
- P**
passive probing, 121
path, source file, 228
PC label
see software addresses
pods, logic analyzer, 287
power on/power off sequence, 114
predefined triggers, 187
preprocessor
See analysis probe
preprocessor interface
See *analysis probe*
problems
inverse assembler, 261
logic analyzer, 261
triggering, 261
processor support package, 119
processors supported, 4
program symbols, 165
- R**
record format, General-Purpose
ASCII, 254
record headers, 252
references, 6
run control tool
See emulation control interface
- S**
Section Format, 252
SECTIONS in GPA format, 256
Setting Up the Logic Analysis
System, 113
setup assistant, 22
-

-
- definition, 287
 - setup checklist, 21
 - show states of type, 223
 - signal availability-impact of, 271
 - signal requirements-for ARM
 - inverse assembly, 42, 90
 - signal-to-connector mappings-ARM, 54, 102
 - signal-to-connector mappings-AMBA, 69
 - skid, reducing, 237
 - slow clock message, 247, 249
 - software
 - installing, 113
 - list of installed, 145
 - requirements, 31
 - software addresses, 227
 - software analyzer, 220, 224
 - software requirements, 31
 - solution
 - definition, 287
 - solutions
 - equipment required, 32
 - product numbers, 4
 - source code
 - displays, 233
 - source correlation, 25
 - using, 225
 - source correlation tool set, 3
 - source file search path, 228
 - SOURCE LINES in GPA format, 259
 - Source Viewer window
 - blank, 225
 - see also source correlation, 225
 - stand-alone logic analyzer
 - definition, 288
 - START ADDRESS in GPA format, 260
 - state analysis, 288
 - definition, 288
 - state symbols, 221, 224
 - store qualify wait states-example, 204, 213
 - subnet mask
 - definition, 288
 - symbol files
 - creating, 252
 - symbols, 163
 - definition, 288
 - object file, 165
 - object module file, 165
 - predefined, 164
 - program, 165
- T**
- target board adapter
 - definition, 288
 - target control port, 288
 - target interface module (TIM)
 - definition, 288
 - target system
 - power sequence, 114
 - The, 22
 - timing analysis, 288
 - definition, 288
 - timing, mode of operation, 153
 - trace
 - missing display, 264
 - transition board
 - definition, 289
 - trigger
 - emulation module, 234
 - on break, 242
 - predefined, 187
 - predefined, viewing, 190
 - sequence, 187
 - source code, 193
 - unwanted, 264
 - trigger function, 191
 - trigger sequence, 192
 - trigger specification
 - definition, 289
 - trigger-example of setting, 198, 206
 - triggering
 - ARM data, 191
 - triggering-ARM data, 195
 - troubleshooting-inverse assembler, 266
- V**
- variable write-16-bit example, 200, 208
 - variable write-example, 207
 - VARIABLES in GPA format, 258
 - versions
 - logic analyzer software, 31
 - viewing-ARM data, 216, 229
- W**
- WAIT signal, 160
 - web sites
 - Agilent logic analyzers, 6
 - See Also under debugger names
 - wizard
 - See setup assistant
-

© Copyright Agilent Technologies
Company 1994-2001
All Rights Reserved.

Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subparagraph (C) (1) (ii) of the Rights in Technical Data and Computer Software Clause in DFARS 252.227-7013.

Agilent Technologies
395 Page Mill Road
Palo Alto, CA 94303-0870 U.S.A.
Rights for non-DOD U.S.
Government Departments and
Agencies are set forth in FAR
52.227-19 (c) (1,2).

Document Warranty

The information contained in this document is subject to change without notice.

Agilent Technologies makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

Agilent Technologies shall not be liable for errors contained herein or for damages in connection with the furnishing, performance, or use of this material.

Safety

This apparatus has been designed and tested in accordance with IEC Publication 1010, Safety Requirements for Measuring Apparatus, and has been supplied in a safe condition. This is a Safety Class I instrument (provided with terminal for protective earthing). Before applying power, verify that the correct safety precautions are taken (see the following warnings). In addition, note the external markings on the instrument that are described under "Safety Symbols."

Warning

- Before turning on the instrument, you must connect the protective earth terminal of the instrument to the protective conductor of the (mains) power cord. The mains plug shall only be inserted in a socket outlet provided with a protective earth contact. You must not negate the protective action by using an extension cord (power cable) without a protective conductor (grounding). Grounding one conductor of a two-conductor outlet is not sufficient protection.
- Only fuses with the required rated current, voltage, and specified type (normal blow, time delay, etc.) should be used. Do not use repaired fuses or short-circuited fuseholders. To do so could cause a shock or fire hazard.

- Service instructions are for trained service personnel. To avoid dangerous electric shock, do not perform any service unless qualified to do so. Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

- If you energize this instrument by an auto transformer (for voltage reduction), make sure the common terminal is connected to the earth terminal of the power source.

- Whenever it is likely that the ground protection is impaired, you must make the instrument inoperative and secure it against any unintended operation.

- Do not operate the instrument in the presence of flammable gasses or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

- Do not install substitute parts or perform any unauthorized modification to the instrument.

- Capacitors inside the instrument may retain a charge even if the instrument is disconnected from its source of supply.

Safety Symbols



Instruction manual symbol: the product is marked with this symbol when it is necessary for you to refer to the instruction manual in order to protect against damage to the product.



Hazardous voltage symbol.



Earth terminal symbol: Used to indicate a circuit common connected to grounded chassis.

WARNING

The Warning sign denotes a hazard. It calls attention to a procedure, practice, or the like, which, if not correctly performed or adhered to, could result in personal injury. Do not proceed beyond a Warning sign until the indicated conditions are fully understood and met.

CAUTION

The Caution sign denotes a hazard. It calls attention to an operating procedure, practice, or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond a Caution symbol until the indicated conditions are fully understood or met.

Product Warranty

This Agilent Technologies product has a warranty against defects in material and workmanship for a period of one year from date of shipment. During the warranty period, Agilent Technologies will, at its option, either repair or replace products that prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Agilent Technologies.

For products returned to Agilent Technologies for warranty service, the Buyer shall prepay shipping charges to Agilent Technologies and Agilent Technologies shall pay shipping charges to return the product to the Buyer. However, the Buyer shall pay all shipping charges, duties, and taxes for products returned to Agilent Technologies from another country.

Agilent Technologies warrants that its software and firmware designated by Agilent Technologies for use with an instrument will execute its programming instructions when properly installed on that instrument. Agilent Technologies does not warrant that the operation of the instrument software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. Agilent Technologies specifically disclaims the implied warranties of merchantability or fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are the buyer's sole and exclusive remedies. Agilent Technologies shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Assistance

Product maintenance agreements and other customer assistance agreements are available for Agilent Technologies products. For any assistance, contact your nearest Agilent Technologies Sales Office.

Certification

Agilent Technologies certifies that this product met its published specifications at the time of shipment from the factory. Agilent Technologies further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology, to the extent allowed by the Institute's calibration facility, and to the calibration facilities of other International Standards Organization members.

About this edition

This is the *Logic Analysis Support for the ARM7/ARM9 User's Guide*.

Publication number
E2493-97007, August 2001
Printed in USA.

The information in this manual previously appeared in
E2493-97006, May 2001
E2493-97005, August 2000
Printed in USA.

Many product updates do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

ARM and ARM7TDMI are registered trademarks of ARM Limited.