

# ARM® CoreSight™ ELA-500 Embedded Logic Analyzer

Revision: r2p1

## Technical Reference Manual



# ARM® CoreSight™ ELA-500 Embedded Logic Analyzer

## Technical Reference Manual

Copyright © 2014-2016 ARM Limited or its affiliates. All rights reserved.

### Release Information

### Document History

Issue	Date	Confidentiality	Change
0000-00-01	30 September 2014	Confidential	First draft for r0p0 at BETA
0000-01	23 January 2015	Confidential	First release for r0p0
0000-02	20 March 2015	Non-Confidential	Second release for r0p0
0100-00	30 October 2015	Non-Confidential	First release for r1p0
0200-00	23 May 2016	Non-Confidential	First release for r2p0
0201-00	09 September 2016	Non-Confidential	First release for r2p1

### Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, ARM makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to ARM’s customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow ARM’s trademark usage guidelines at <http://www.arm.com/about/trademark-usage-guidelines.php>

Copyright © 2014-2016, ARM Limited or its affiliates. All rights reserved.

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

**Confidentiality Status**

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

**Product Status**

The information in this document is Final, that is for a developed product.

**Web Address**

<http://www.arm.com>

# Contents

## ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Technical Reference Manual

### **Preface**

<i>About this book</i> .....	7
<i>Feedback</i> .....	10

### **Chapter 1**

#### **Introduction**

1.1	<i>About the ELA-500 Embedded Logic Analyzer</i> .....	1-12
1.2	<i>Definitions of terms used in this book</i> .....	1-13
1.3	<i>Compliance</i> .....	1-14
1.4	<i>Features</i> .....	1-15
1.5	<i>Interfaces</i> .....	1-16
1.6	<i>Configuration options</i> .....	1-17
1.7	<i>Test features</i> .....	1-18
1.8	<i>Product documentation and design flow</i> .....	1-19
1.9	<i>Product revisions</i> .....	1-21

### **Chapter 2**

#### **Functional description**

2.1	<i>About the functions</i> .....	2-23
2.2	<i>Interfaces</i> .....	2-25
2.3	<i>Clocking and reset</i> .....	2-27
2.4	<i>Trace control and capture</i> .....	2-29
2.5	<i>Triggering</i> .....	2-33
2.6	<i>Authentication interface</i> .....	2-38

2.7	Parameter summary .....	2-39
-----	-------------------------	------

## Chapter 3

### Programmers model

3.1	Access permissions .....	3-42
3.2	Programming sequence .....	3-43
3.3	Control register summary .....	3-44
3.4	Control register descriptions .....	3-45
3.5	Current State register summary .....	3-48
3.6	Current State register descriptions .....	3-49
3.7	RAM register summary .....	3-51
3.8	RAM register descriptions .....	3-52
3.9	Trigger State register summary .....	3-55
3.10	Trigger State register descriptions .....	3-58
3.11	Integration Mode register summary .....	3-67
3.12	Integration Mode register descriptions .....	3-68
3.13	Software Lock register summary .....	3-70
3.14	Software Lock register descriptions .....	3-71
3.15	Authentication register summary .....	3-72
3.16	Authentication register descriptions .....	3-73
3.17	Device register summary .....	3-74
3.18	Device register descriptions .....	3-75
3.19	ID register summary .....	3-78
3.20	ID register descriptions .....	3-79

## Appendix A

### Signal descriptions

A.1	Clocks and reset .....	Appx-A-84
A.2	Debug APB signals .....	Appx-A-85
A.3	Observation interface signals .....	Appx-A-86
A.4	Timestamp interface signals .....	Appx-A-87
A.5	Authentication interface signals .....	Appx-A-88
A.6	DFT and MBIST interface signals .....	Appx-A-89
A.7	Q-Channel Low-Power interface signals .....	Appx-A-90
A.8	Output Action signals .....	Appx-A-91
A.9	External Trigger Input signals .....	Appx-A-92

## Appendix B

### Revisions

B.1	Revisions .....	Appx-B-94
-----	-----------------	-----------

# Preface

This preface introduces the *ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Technical Reference Manual*.

It contains the following:

- [About this book](#) on page 7.
- [Feedback](#) on page 10.

## About this book

This book is for the ARM® CoreSight ELA-500 Embedded Logic Analyzer.

### Product revision status

The *rm**pn* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

*rm* Identifies the major revision of the product, for example, r1.

*pn* Identifies the minor revision or modification status of the product, for example, p2.

### Intended audience

This book is written for system designers, system integrators, and programmers who are designing or programming a *System-on-Chip* (SoC) that uses the ELA-500.

### Using this book

This book is organized into the following chapters:

#### **Chapter 1 Introduction**

This chapter describes the ELA-500 Embedded Logic Analyzer.

#### **Chapter 2 Functional description**

This chapter describes the functionality of the ELA-500.

#### **Chapter 3 Programmers model**

This chapter describes the programmers model.

#### **Appendix A Signal descriptions**

This appendix describes the external signals of the ELA-500.

#### **Appendix B Revisions**

This appendix describes the technical changes between released issues of this book.

### Glossary

The ARM Glossary is a list of terms used in ARM documentation, together with definitions for those terms. The ARM Glossary does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See the [ARM Glossary](#) for more information.

### Typographic conventions

*italic*

Introduces special terminology, denotes cross-references, and citations.

**bold**

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

monospace

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

monospace

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

*monospace italic*

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

**monospace bold**

Denotes language keywords when used outside example code.

<and>

Encloses replaceable terms for assembler syntax where they appear in code or code fragments.  
For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

## Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

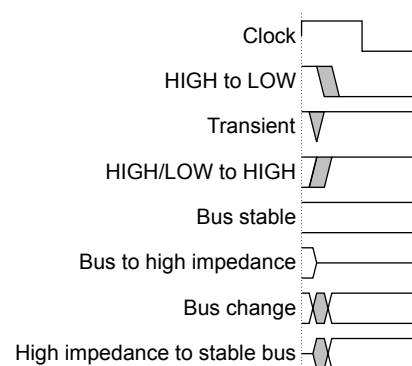


Figure 1 Key to timing diagram conventions

## Signals

The signal conventions are:

### Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW.  
Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

### Lowercase n

At the start or end of a signal name denotes an active-LOW signal.

## Additional reading

This section lists publications by ARM and by third parties.

See *Infocenter* <http://infocenter.arm.com>, for access to ARM documentation.

### ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *ARM® Low Power Interface Specification, Q-Channel and P-Channel Interfaces* (ARM IHI 0068).
- *ARM® AMBA® AXI and ACE Protocol Specification* (ARM IHI 0022).
- *ARM® AMBA® APB Protocol Specification* (ARM IHI 0024).
- *ARM® CoreSight™ Architecture Specification* (ARM IHI 0029).



The following confidential books are only available to licensees:

- *ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Integration and Implementation Manual* (ARM 100129).

#### **Other publications**

This section lists relevant documents published by third parties:

- *JEDEC Standard Manufacturer's Identification Code, JEP106* <http://www.jedec.org>.

## Feedback

### Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

### Feedback on content

If you have comments on content then send an e-mail to [errata@arm.com](mailto:errata@arm.com). Give:

- The title *ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Technical Reference Manual*.
- The number ARM 100127\_0201\_00\_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

————— **Note** —————

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

---

# Chapter 1

## Introduction

This chapter describes the ELA-500 Embedded Logic Analyzer.

It contains the following sections:

- *1.1 About the ELA-500 Embedded Logic Analyzer* on page 1-12.
- *1.2 Definitions of terms used in this book* on page 1-13.
- *1.3 Compliance* on page 1-14.
- *1.4 Features* on page 1-15.
- *1.5 Interfaces* on page 1-16.
- *1.6 Configuration options* on page 1-17.
- *1.7 Test features* on page 1-18.
- *1.8 Product documentation and design flow* on page 1-19.
- *1.9 Product revisions* on page 1-21.

## 1.1 About the ELA-500 Embedded Logic Analyzer

The ELA-500 Embedded Logic Analyzer is a component for debugging hardware-related issues.

Debug signals are connected from the IP being debugged to the ELA-500, which compares the signals with a target value and drives actions. There is an optional trace capability that can be used to generate a history of the debug signals at any point in time.

## 1.2 Definitions of terms used in this book

This Technical Reference Manual uses terms that are specific to the ELA-500 Embedded Logic Analyzer.

The following terms have specific meanings within the context of this Technical Reference Manual. Wherever they are used throughout the book they are shown in *italics* and have the meanings that are shown here:

### Trigger State

One of the five states that the ELA-500 trigger logic can be in. The *Trigger State* controls which *Signal Group* signals are routed to the comparison logic, target comparison values, comparison and counter control, and output actions. The ELA-500 advances to the next *Trigger State* when its *Trigger Condition* is met.

#### ————— Note —————

The sequence of *Trigger States* is programmable and does not depend on implementation.

### Trigger Signal Comparison

The comparison of the *External Trigger Input Signals* and selected *Signal Group* with a target value and mask that is determined by the current *Trigger State*.

### Trigger Counter Comparison

The comparison of the up-counter of the current *Trigger State* with its target value. The counter can be incremented by **ELACLK** or by *Trigger Signal Comparison* matches. The counter can be reset by a *Trigger Signal Comparison* match.

### Trigger Signal Alternative Comparison

An alternative comparison of the *External Trigger Input Signals* and selected *Signal Group* with a target value and mask that is determined by the current *Trigger State*.

### Trigger Condition

When the *Trigger Condition* is met, the ELA-500 generates an *Output Action* and transitions to the next *Trigger State*. If *Trigger Counter Comparison* is enabled, the *Trigger Condition* is met when the *Trigger Counter Comparison* is true. If *Trigger Counter Comparison* is disabled, the *Trigger Condition* is met when the *Trigger Signal Comparison* is met.

### External Trigger Input Signals

The ELA-500 supports eight input signals that can form part of the *Trigger Signal Comparison*. The *External Trigger Input Signals* can come from other ELA-500 instances, a CoreSight Cross Trigger Interface, or other logic in the SoC.

### Signal Group

A group of input signals from the Observation interface. The ELA-500 supports up to 12 *Signal Groups*, each of which is 64 bits, 128 bits, or 256 bits wide, determined by the **GRP\_WIDTH** parameter.

### Output Action

The ELA-500 generates an *Output Action* when the *Trigger Condition* is met.

The *Output Action* can:

- Drive the **STOPCLOCK** output for scan-dump analysis.
- Drive a CoreSight Embedded Cross Trigger through **CTTRIGOUT[1:0]** to a CoreSight *Cross Trigger Interface* (CTI).
- Drive other logic through **ELAOUTPUT[3:0]**.

## 1.3 Compliance

The ELA-500 implements the ARM CoreSight Architecture Specification. It complies with the AMBA APB Protocol and the ARM Low Power interface Q-Channel specification.

This Technical Reference Manual complements architecture reference manuals, architecture specifications, protocol specifications, and relevant external standards. It does not duplicate information from these sources.

See the *ARM® AMBA® APB Protocol Specification*, the *ARM® CoreSight™ Architecture Specification*, and the *ARM® Low Power Interface Specification, Q-Channel and P-Channel Interfaces* for more information.

## 1.4 Features

The ELA-500 has many features. Some of the key features are programmable *Trigger States*, programmable *Output Actions* for each *Trigger State*, and additional *External Trigger Input Signals*.

The ELA-500 has the following key features:

- Four programmable *Trigger States*.
- Eight programmable actions, to allow each *Trigger State* to control:
  - Stop clock.
  - Trace control.
  - CoreSight cross-trigger.
  - Four general-purpose trigger outputs.
- A programmable 32-bit counter for each *Trigger State* that can be used to delay output actions, count events, or as a watchdog timer.
- An Observation interface consisting of 12 *Signal Groups* with a configurable width of 64, 128, or 256 debug signals.
- Eight *External Trigger Input Signals* that can be masked and compared against a target value for each *Trigger State*. An *Output Action* from one logic analyzer can be connected to these inputs on a second logic analyzer to cause a direct cross-trigger, independently of the CoreSight *Embedded Cross Trigger* (ECT) infrastructure. This feature enables users to have a lower-latency *Embedded Logic Analyzer* (ELA) cross-trigger mechanism that does not rely on the correct operation of software-debug cross-triggering components.
- Programmable *Trigger Condition* comparison with the ability to change the target comparison by selectively masking signals, and selecting =, !=, <, <=, >, >= for comparison of the masked signals and counter target value comparisons.
- Programmable *Trigger Alternative Condition* comparison with the ability to change the target comparison by selectively masking signals, and selecting =, !=, <, <=, >, >=, for comparison of the masked signals and counter target value comparisons.
- Optional support of signal trace using an integrated SRAM. The SRAM trace depth is configurable. Timestamp trace capture is possible and enables correlation of ELA trace with other CoreSight trace sources.

## 1.5 Interfaces

The ELA-500 has numerous external interfaces, including interfaces for debug signals, trigger inputs, and authentication permissions.

The ELA-500 has the following external interfaces:

- An Observation interface to capture signals from the IP being debugged.
- An *External Trigger Input Signals* interface that enables the ELA-500 to be triggered by external logic.
- An Authentication interface that determines the type of accesses permitted.
- A debug APB slave interface that enables access to the configuration and status registers.
- An SRAM interface to enable access to SRAM for trace data capture.
- A timestamp interface to provide timestamp information with captured trace data.
- A Low-Power Q-Channel interface to determine when **ELACLK** can be stopped.
- A *Memory Built-In Self-Test* (MBIST) interface for testing SRAM.



## 1.6 Configuration options

This section describes the configuration options available in the ELA-500.

### 1.6.1 Configurable parameters

There are several configurable options available in the ELA-500.

#### **Related concepts**

[2.7 Parameter summary](#) on page 2-39.

### 1.6.2 Static parameters

There are no configurable static parameters in the ELA-500.

### 1.6.3 Tie-off signals

There are no configurable tie-off signals in the ELA-500.

## 1.7 Test features

The ELA-500 has several test features.

See the *ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Integration and Implementation Manual* for information about the test features.

## 1.8 Product documentation and design flow

The ELA-500 documentation includes a Technical Reference Manual and an Integration and Implementation Manual. These books relate to the ELA-500 design flow.

### Documentation

The ELA-500 documentation includes the following books:

#### Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the ELA-500. It is required at all stages of the design flow. The choices that you make in the design flow can mean that some behavior described in the TRM is not relevant. If you are programming the ELA-500 then contact:

- The implementer to determine:
  - What integration, if any, was performed before implementing the ELA-500.
  - The build configuration of the implementation.

#### Note

Build configuration information is also readable from the DEVID registers.

- The integrator to determine the pin configuration of the device that you are using.

#### Integration and Implementation Manual

The *Integration and Implementation Manual* (IIM) describes:

- The available build configuration options and related issues in selecting them.
- How to configure the *Register Transfer Level* (RTL) with the build configuration options.
- How to integrate the ELA-500 into a SoC. This includes a description of the integration kit and describes the pins that the integrator must tie off to configure the macrocell for the required integration.
- How to implement the ELA-500 into your design. This includes floorplanning guidelines, *Design for Test* (DFT) information, and how to perform netlist dynamic verification on the ELA-500.
- The processes to sign off the integration and implementation of the design.

The ARM product deliverables include reference scripts and information about using them to implement your design.

Reference methodology documentation from your EDA tools vendor complements the IIM.

The IIM is a confidential book that is only available to licensees.

### Design flow

The ELA-500 is delivered as synthesizable RTL. Before it can be used in a product, it must go through the following processes:

#### Implementation

The implementer configures and synthesizes the RTL to produce a hard macrocell. This includes integrating RAMs into the design.

#### Integration

The integrator connects the implemented design into a SoC. This includes connecting it to a memory system and peripherals.

#### Programming

This is the final process. The system programmer develops the software that is required to configure and initialize the ELA-500, and tests the required application software.

Each process:

- Can be performed by a different party.
- Can include implementation and integration choices that affect the behavior and features of the ELA-500.

The operation of the final device depends on:

**Build configuration**

The implementer chooses the options that affect how the RTL source files are pre-processed. These options usually include or exclude logic that affects one or more of the area, maximum frequency, and features of the resulting macrocell.

**Software configuration**

The programmer configures the ELA-500 by programming particular values into registers. This affects the behavior of the ELA-500.

---

**Note**

This Technical Reference Manual refers to implementation-defined features that are applicable to build configuration options. Reference to a feature that is included means that the appropriate build and pin configuration options are selected. Reference to an enabled feature means a feature that has been configured by software.

---

## 1.9 Product revisions

This section describes the differences in functionality between product revisions of the ELA-500.

**r0p0** First release.

**r1p0** Adds new features to support the interconnect. These features are disabled by default with parameters. The ELA-500 is backward compatible with revision r0p0.

The new features are:

- Conditional *Trigger States*.
- Transaction ID capture for use in a subsequent *Trigger State* comparison.
- Addition of a fifth *Trigger State* - *Trigger State 4*.
- *Trigger State 4* can run a separate comparator and trace in a separate loop to the other four *Trigger States*. Trace from *Trigger State 4* dominates trace writes that occur at the same time from any of the other four *Trigger States*.

Other changes in r1p0 are:

- DEVID2 and PID2 registers added.
- CTSR bit 31 FINALSTATE added.

**r2p0** Adds the following new features:

- Support for 256-bit **SIGNALGRPs** and trace.
- A trace read data scrambler to protect exposure of designs that are sensitive to signal trace.
- The trace read data scrambler has added new parameters.
- The new TRIGIN\_EDGE parameter supports a single rising-edge trigger inputs.
- A single rising-edge trigger configuration for **CTTRIGIN** and **EXTRIGIN** cross trigger inputs.
- The Peripheral ID2 register [7:4] REVISION field was changed from 0x1 to 0x2 to identify r2p0.

**r2p1** Adds the following features:

- Synchronizers replaced with a cell model.
- Peripheral ID2 register bits [7:4] REVISION field changed from 0x2 to 0x3 to identify r2p1.

# Chapter 2

## Functional description

This chapter describes the functionality of the ELA-500.

It contains the following sections:

- [2.1 About the functions](#) on page 2-23.
- [2.2 Interfaces](#) on page 2-25.
- [2.3 Clocking and reset](#) on page 2-27.
- [2.4 Trace control and capture](#) on page 2-29.
- [2.5 Triggering](#) on page 2-33.
- [2.6 Authentication interface](#) on page 2-38.
- [2.7 Parameter summary](#) on page 2-39.

## 2.1 About the functions

This section describes the functional blocks in the ELA-500.

The ELA-500 can be configured to have 64, 128, or 256 debug signals in a *Signal Group*. There are eight *External Trigger Input Signals* that can be used for cross-triggering from other CoreSight components including another ELA-500.

The ELA-500 is programmed from an APB bus and has architectural registers that enable identification in the CoreSight topology.

The ELA-500 also provides support for:

- A CoreSight authentication interface.
- An optional SRAM trace unit with configurable trace depth.
- Insertion of timestamps into the trace data.

There are seven output actions that can be used for various functions, such as stopping the clock to enable the system state to be extracted using a scan chain, cross triggering to a CoreSight debug subsystem, and other system-specific actions.

The following figure shows the functional blocks of the ELA-500:

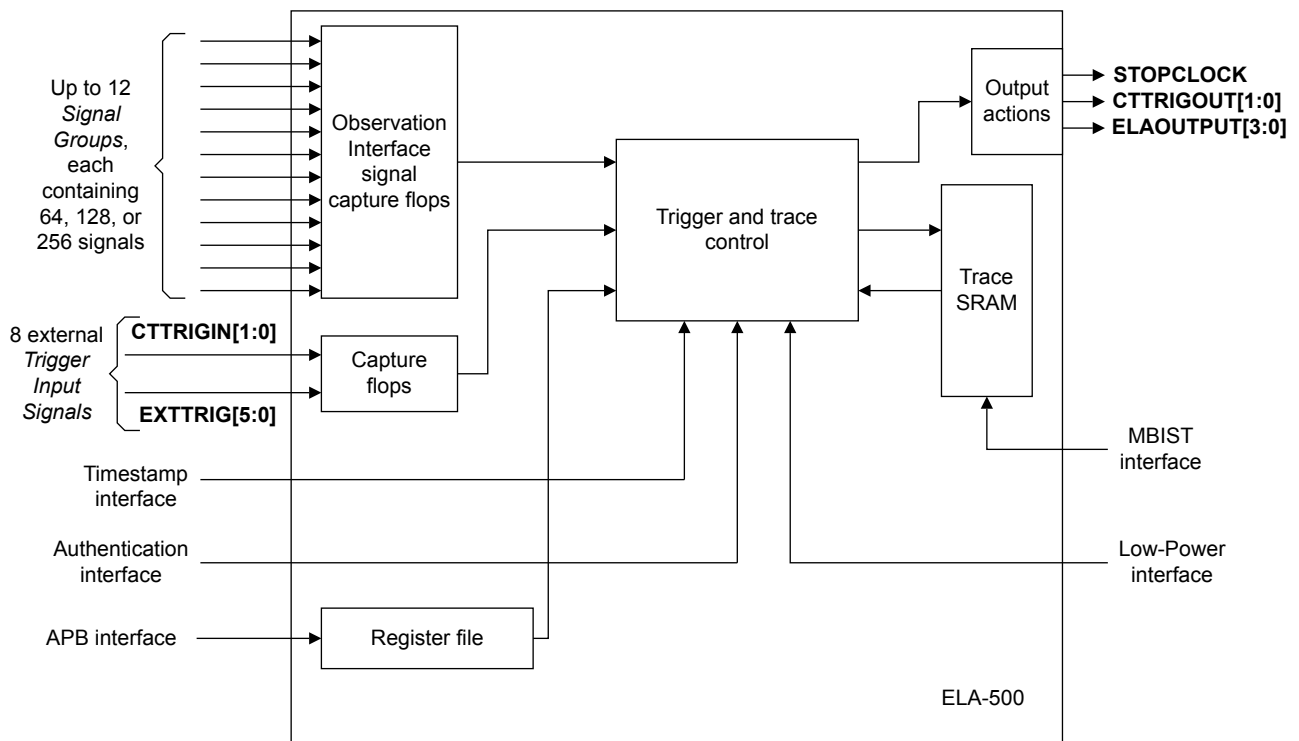


Figure 2-1 ELA-500 block diagram

The following figure shows how to use an ELA-500 in a system:

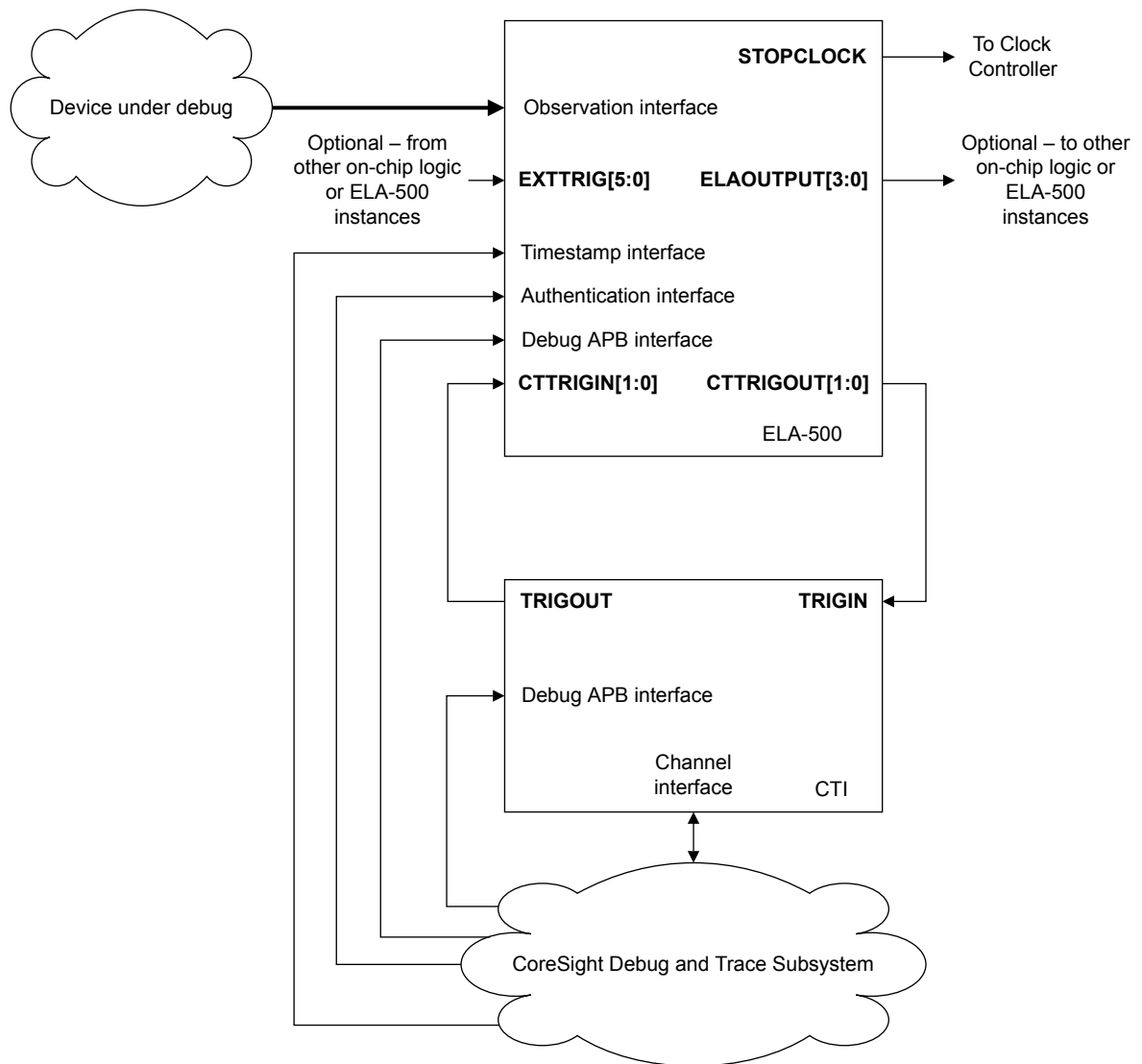


Figure 2-2 How to use the ELA-500 in a system



## 2.2 Interfaces

The ELA-500 has numerous external interfaces. Some of these include interfaces to debug signals, trigger inputs, and authentication permissions.

The ELA-500 has the following interfaces:

### Debug APB slave interface

This interface provides access to the ELA-500 configuration register and status registers. See the *ARM® AMBA® APB Protocol Specification* and the *ARM® CoreSight™ Architecture Specification* for more information about the debug APB signals.

### Observation Interface

This consists of 12 *Signal Group* buses of 64, 128, or 256 bits, depending on the configuration parameter GRP\_WIDTH.

### External Trigger inputs

There are eight trigger inputs that can be used as trigger conditions. These inputs can be sourced from signals from a CoreSight CTI, or other on-chip signals, such as interrupts and debug requests, or can be an output signal from another ELA-500.

### Timestamp interface

This interface accepts a 64-bit natural binary value from a timestamp generator in the system. The timestamp is captured alongside trace data in the SRAM.

### Authentication interface

The **DBGEN**, **NIDEN**, **SPIDEN**, and **SPNIDEN** signals are supported as described in the *ARM® CoreSight™ Architecture Specification*.

### SRAM trace interface

If present, the SRAM trace interface connects to the SRAM that is used to store the captured trace data.

The following figure shows the SRAM read access timing:

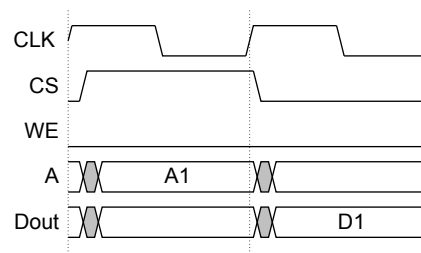


Figure 2-3 SRAM read access timing

The following figure shows the SRAM write access timing:

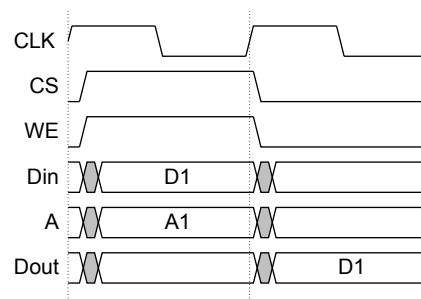


Figure 2-4 SRAM write access timing

### SRAM MBIST interface

The *Memory Built-In Self-Test* (MBIST) interface provides a functional access path to the memories for self-test purposes.

### Low-Power interface Q-Channel

The ELA-500 provides a *Low-Power Interface* (LPI) that can be used by a clock controller to determine whether **ELACLK** can be stopped. The ELA-500 can be stopped when the following conditions are met:

- CTRL.RUN = 0.
- There are no pending debug APB register accesses, that is, **PSELDBG** is LOW.
- There is no SRAM access in progress.
- Integration test mode is disabled with ITCTRL.IME = 0.

The **ELAQACTIVE** signal is driven by an OR of the following signals:

- Unsynchronized **PSELDBG**.
- SRAM busy, that indicates a one cycle read or write is in progress.
- CTRL.RUN.
- ITCTRL.IME.

ITCTRL.IME = 1 is used to assert **ELAQACTIVE** so that **ELACLK** continues to run when integration mode is enabled, to avoid clock gating during integration test of the pulsed signals on **CTTRIGIN** and **EXTTRIGIN**.

An internal active signal is generated with synchronized **PSELDBG**. This internal active signal is used when **ELAQREQn** requests are asserted, to move to the Q\_STOPPED state when internal active is LOW, or to the Q\_DENIED state when internal active is HIGH.

### Related references

[A.6 DFT and MBIST interface signals on page Appx-A-89.](#)

[A.7 Q-Channel Low-Power interface signals on page Appx-A-90.](#)

## 2.3 Clocking and reset

This section describes the clock and reset signals and procedures for the ELA-500.

This section contains the following subsections:

- [2.3.1 Clocking on page 2-27](#).
- [2.3.2 Reset on page 2-28](#).

### 2.3.1 Clocking

The ELA-500 has two main clock domains, the **PCLKDBG** domain that contains the debug APB interface, and the **ELACLK** domain that is the clock for sampling on the Observation interface, for *Signal Groups*, *Trigger State* comparisons, counters, and output actions.

Using multiple clock domains enables you to run the debug APB at much slower frequencies than sampled IP, such as cores, that can run at above 2GHz.

The following figure shows the division of the clock domains within the ELA-500.

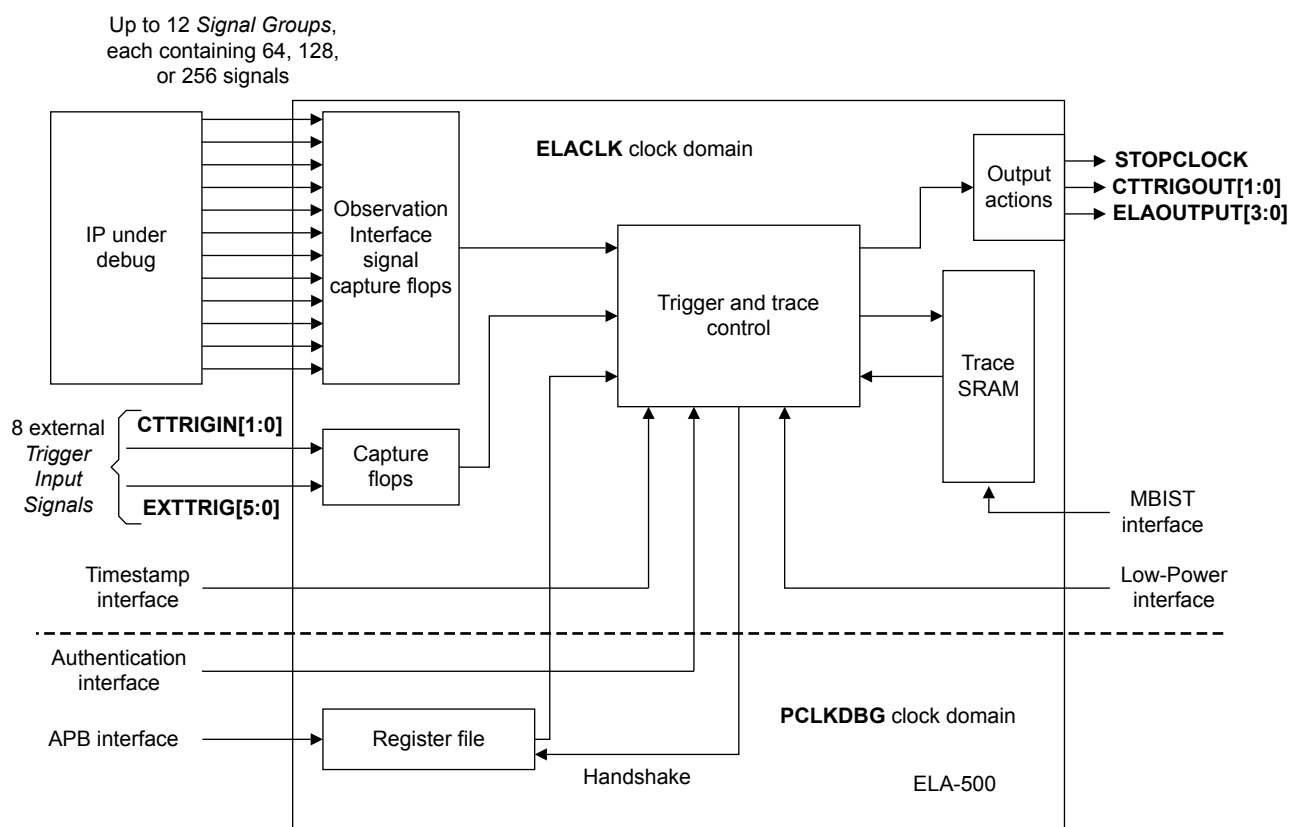


Figure 2-5 ELA-500 clock domains

#### Clock domain synchronization

The ELA-500 performs synchronization of some input signals, and some signals between the **PCLKDBG** and **ELACLK** domains.

Software must take into account the following restrictions:

- The debug APB registers must only be written when the ELA-500 is stopped, that is, when CTRL.RUN is low.
- The debug APB registers must only be read when the ELA-500 is stopped. The exceptions to this rule are the Current State registers that can be read at any time to inspect the current state.

When implementing the ELA-500 in a system, you must be aware of the following points:

- All the debug signals in the Observation interface are sampled by **ELACLK**. These signals are not synchronized to **ELACLK** inside the ELA-500, and must be driven from logic clocked by **ELACLK** outside the ELA-500.
- The Authentication interface signals, **SPIDEN**, **DBGEN**, **NIDEN**, and **SPNIDEN**, must be synchronized to **PCLKDBG** outside the ELA-500.
- To aid debug, it is advantageous to have the logic analyzer operational during reset or during power management events of the sampled IP, such as powering down one of multiple cores. Take care with debug signals that originate from processors or other IP. The sampled IP could be power-gated or have asynchronous resets that could cause glitching or false sampling by the logic analyzer. De-assert **SIGCLKEN<n>** when debug inputs are changing because of:
  - Assertion or de-assertion of isolation logic for debug signals that cross power domains.
  - Assertion or de-assertion of asynchronous resets to the sampled IP.
- The **STOPCLOCK** output must not affect the initialization of the SoC following a powerup reset.
- The **TSVALUE** signals in the Timestamp interface must be synchronized to **ELACLK**.

### Related references

[3.6 Current State register descriptions on page 3-49.](#)

## 2.3.2 Reset

The ELA-500 has two functional resets, each corresponding to one of the two clock domains in the ELA-500.

The resets are:

- **RESETn** that resets the logic in the **ELACLK** domain.
- **PRESETDBGn** that resets the logic in the **PCLKDBG** domain.

Both resets can be asserted asynchronously to the respective clock, and must be synchronously de-asserted. **RESETn** must only be asserted when powering up the **ELACLK** domain. **PRESETDBGn** must be asserted when powering up the **PCLKDBG** domain, or when a reset of system debug logic is required. Assertion of **PRESETDBGn** when the CTRL.RUN bit is set disables the ELA-500.

#### Note

ARM recommends that **PRESETDBGn** is only asserted when the ELA-500 is already disabled with CTRL.RUN set to 0b0.

A third reset signal, **nMBISTRESET**, is an asynchronous test mode reset for both **ELACLK** and **PCLKDBG** domains. **nMBISTRESET** must be driven HIGH for functional operation and reset. See the *ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Integration and Implementation Manual* for more information on test mode.

## 2.4 Trace control and capture

The SRAM trace unit on the ELA-500 is configurable using the parameters `RAM_ADDR_SIZE` and `TRACE_GEN`.

The value of `RAM_ADDR_SIZE` represents the number of SRAM address bits. Software can read the `DEVID` register to determine trace depth. The trace SRAM acts as a circular buffer when trace is being captured, with the SRAM address incrementing automatically.

This section contains the following subsections:

- [2.4.1 Trace control on page 2-29.](#)
- [2.4.2 Trace capture on page 2-29.](#)
- [2.4.3 Second trace comparator on Trigger State 4 on page 2-29.](#)
- [2.4.4 Trace SRAM format on page 2-30.](#)
- [2.4.5 Timestamp control on page 2-31.](#)
- [2.4.6 Debug APB registers and interface to SRAM on page 2-31.](#)

### 2.4.1 Trace control

Trace is controlled by the `CTRL`, `TRIGCTRL`, `PTACTION`, and `ACTION<n>` registers.

Trace can only be active when the ELA-500 is running, that is when `CTRL.RUN = 1`. If `PTACTION.TRACE` is set, trace becomes active when `CTRL.RUN` is set.

When the ELA-500 is running, trace is controlled by `ACTION<n>.TRACE`. It is therefore possible to enable or disable trace at each *Trigger State* transition.

### 2.4.2 Trace capture

When trace is active, trace capture is controlled in each *Trigger State* by the *Trigger Control Register* (`TRIGCTL<n>`).

The following trace capture options are available:

- Capture on every **ELACLK**.
- Capture on a *Trigger Signal Comparison* match.
- Capture on a *Trigger Counter Comparison* match.

#### Related references

[3.10.2 Trigger Control registers on page 3-59.](#)

### 2.4.3 Second trace comparator on Trigger State 4

*Trigger State 4* includes additional capability that allows *Trigger State 4* to trace `SIGNALGRP<n>` data while the other four *Trigger States* are programmed for comparisons and trace writes. *Trigger State 4* is generated by setting the `NUM_TRIG_STATES` parameter to 5.

If multiple trace writes for *Trigger States* occur on the same clock cycle, trace from the highest numbered *Trigger State* takes priority. The trace write from the lower numbered *Trigger States* is dropped and the trace data overwrite bit, `bit[5]`, in the trace header byte is set. In this implementation, where `DEVID2 = 4`, the trace data from *Trigger State 4* will take precedence when there is a simultaneous write from another *Trigger State*. This feature allows a prioritized trace of two **SIGNALGRPs** at the same time.

The *Trigger State Select Register* (`TSSR`), controls the second trace comparator. Setting `bit[4]`, that is `ALTTS[4] = 1`, enables the independent trace capability of *Trigger State 4*. When this bit is set, *Trigger State 4* cannot be used in a loop with other *Trigger States* using `NEXTSTATE<n>` and `ALTNEXTSTATE<n>` with *Trigger State 4* as a destination. Also, *Trigger State 4* cannot update or drive an `ACTION` or `ALTACTION`.

*Trigger State 4* has the following characteristics:

- Alternative comparisons do not function.
- *Trigger State 4* runs in a continuous trace loop as if the NEXTSTATE is programmed to go back to *Trigger State 4*.
- *Trigger State 4* trace stops when the other *Trigger States* reach a final state or CTRL.RUN is set to 0.
- The counters and trace filtering options are functional for *Trigger State 4*. TRIGCTRL4.COMPSRC, TRIGCTRL4.WATCHRST, TRIGCTRL4.COUNTSRC, TRIGCTRL4.TRACE, and TRIGCTRL4.COUNTCLR are available when ALTTS4 = 1. TRIGCTRL4.CAPTID, TRIGCTRL4.ALTCOMP, TRIGCTRL4.COUNTBRK, and TRIGCTRL4.ALTCOMPSRC are not available.
- Loops that are based on counter reset add an extra clock for the reset. For example, setting COUNTCOMP4 = 5 with TRIGCTRL4.COUNTSRC = 1, TRIGCTRL4.TRACESRC = 1 and TRIGCTRL4.COMP = 3'b001, results in trace capture when the counter reaches five trigger signal comparisons. The trace capture stays asserted until final\_state is reached. Setting TRIGCTRL4.COUNTCLR = 1 resets the counter after five trigger signal comparisons are counted. A trace is captured after every five trigger signal comparisons, with the *Trigger State* going to final\_state one **ELACK** cycle later, or if CTRL.RUN is cleared. If TRIGCTRL4.TRACESRC = 0, a trace is captured every six **ELACK** cycles when COUNTCOMP4 = 5.

#### 2.4.4 Trace SRAM format

The trace SRAM can capture a full *Signal Group* on every **ELACK** cycle. The width of the trace SRAM is GRP\_WIDTH + 8, with the additional eight bits being used to record a header byte that identifies the data as either a timestamp or a capture of the *Signal Group*.

Each trace SRAM word contains a data payload and a header byte. The header byte is located at the least significant byte of the SRAM word. The payload data is located in the upper bytes of the SRAM word.

For example, for a 64-bit *Signal Group* configuration:

```
GRP_WIDTH = 64
SRAM data[71:0] = {payload[63:0], header[7:0]}
```

For a 128-bit *Signal Group* configuration:

```
GRP_WIDTH = 128
SRAM data[135:0] = {payload[127:0], header[7:0]}
```

For a 256-bit *Signal Group* configuration:

```
GRP_WIDTH = 256
SRAM data[263:0] = {payload[255:0], header[7:0]}
```

The following table shows the header byte format.

**Table 2-1 Header byte format**

Bits	Name	Function
[7:6]	Trace counter[1:0]	Two selected bits from a 16-bit cycle counter in the trace unit, used to identify a fine-grain time associated with the trace capture. The Timestamp Control Register controls the selection of the counter bits that are used.
[5]	Trace data overwrite	Identifies that <i>Trigger State 4</i> has overwritten data that was being written at the same time from a different <i>Trigger State</i> , that is <i>Trigger States</i> 0-3. Requires that TSSR.ALTTS4 = 1.

Table 2-1 Header byte format (continued)

Bits	Name	Function
[4:2]	Trigger state	Current <i>Trigger State</i> . Software can use the <i>Trigger State</i> to determine which <b>SIGNALGRP&lt;n&gt;</b> was traced, by reading SIGSEL<n>.
[1:0]	Type	<p>Returns the type of data that follows the header byte:</p> <p><b>0b00</b>      Reserved.</p> <p><b>0b01</b>      A 64-bit, 128-bit, or 256-bit data payload follows the header.</p> <p><b>0b10</b>      A timestamp value follows the header.</p> <p style="text-align: center;">————— <b>Note</b> —————</p> <p>Timestamps are optional and can be enabled using the Timestamp Control Register. A timestamp payload contains the full 64-bit timestamp value. If the ELA-500 is configured with GRP_WIDTH &gt; 64, the payload is zero-extended above the 64-bit timestamp value.</p> <p><b>0b11</b>      Reserved.</p>

**Related references**

[3.4.2 Timestamp Control register on page 3-45.](#)

**2.4.5 Timestamp control**

Timestamps enable correlation of ELA-500 trace with trace from other CoreSight trace sources.

Timestamps in the ELA-500 have the following features:

- The Timestamp Control Register is used to enable writing of timestamps into the trace SRAM.
- Timestamps plus the associated header byte occupy 72, 136, or 264 bits of data. If the ELA-500 is configured with GRP\_WIDTH > 64, the timestamp value is zero-extended to 128 or 256 bits.
- Trace filtering that does not capture debug signal data on every **ELACLK** cycle enables timestamps to be written into the trace SRAM based on the interval set in the Timestamp Control Register. When the programmed timestamp interval is reached, a request is generated to insert a timestamp in the next available cycle that does not have a debug signal trace capture.
- Timestamps can be written into the trace SRAM after the trace active action is de-asserted when TIMECTRL.TSINT = 0. This guarantees that at least one timestamp is present in the circular SRAM buffer.
- When CTRL.RUN is cleared, a timestamp is written into trace SRAM if the previous trace write contained a data payload.

**2.4.6 Debug APB registers and interface to SRAM**

When configured with TRACE\_GEN = 1, the SRAM is accessible through the debug APB registers.

The SRAM is 72 bits, 136 bits, or 264 bits wide, depending on the *Signal Group* width configuration parameter GRP\_WIDTH.

Four registers enable the SRAM to be accessed through the 32-bit debug APB interface:

- *RAM Read Address Register* (RRAR).
- *RAM Read Data Register* (RRDR).
- *RAM Write Address Register* (RWAR).
- *RAM Write Data Register* (RWDR).

The RAM Read registers are provided to enable a debugger to read out captured trace data from the ELA-500. The RAM Write registers are provided to support integration testing.

The RRAR and RWAR address single 72-bit, 136-bit, or 264-bit words within the SRAM. Multiple RRDR or RWDR accesses are required for each SRAM word. An internal holding register is used to transfer data between the SRAM and RAM Data registers.

## SRAM reads

When the RRAR is updated, either by a debug APB write or by an automatic increment, the SRAM data at that address is copied to the holding register.

Reads to the RRDR return the data from the holding register. The first read of the RRDR after an RRAR update returns the trace data header byte value, zero-extended to 32-bits. Subsequent reads of the RRDR return 32-bit chunks of the trace data payload, starting with the least significant chunk. This continues until all the payload data has been read, that is, two chunks if GRP\_WIDTH = 64, four chunks if GRP\_WIDTH = 128, and eight chunks if GRP\_WIDTH = 256.

When the final 32 bits of the payload have been read, the RRAR is incremented automatically. Then, the next word of SRAM data is copied into the holding register. This enables the SRAM data content to be read out efficiently.

The RRAR wraps to address zero if it is incremented beyond the maximum depth of the SRAM.

The trace read data scrambler protects the exposure of designs that are sensitive to signal trace. One or more signal groups might be scrambled. You must know the details of the specific scrambling system that was used to be able to unscramble the trace data.

For more information, see *ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Integration and Implementation Manual*.

## SRAM writes

Writes to the SRAM are supported for integration-testing purposes.

A write to the RWAR sets the SRAM address for the data that is then written to the RWDR. Writes to the RWDR update the internal holding register.

The first write to the RWDR sets the header byte value with the least significant byte written. Subsequent writes to the RWDR set 32-bit chunks of the payload, starting with the least significant chunk. When the final 32 bits of the payload have been written, the content of the holding register is copied into the SRAM and the RWAR is incremented automatically.

The RWAR wraps to address zero if it is incremented beyond the maximum depth of the SRAM.



## 2.5 Triggering

Triggering is the process of causing an *Output Action* signal to be driven, or advancing to the next *Trigger State*, when a *Trigger Signal Comparison* or a *Trigger Counter Comparison* match occurs.

*Trigger Signal Comparisons* are based on the comparison of *Signal Groups* and *External Trigger Input Signals*. The masked *Signal Group* values and target values in the *Signal Compare* (SIGCOMP<n>) registers are compared, and logically ANDed, with the comparison of the masked *External Trigger Input Signals* and target values in the *External Compare* register (EXTCOMP<n>), where  $n = 0-4$  and denotes one of the five *Trigger States*. The *Trigger Condition* is only met if both the *Signal Group* and *External Trigger Input Signal* comparisons succeed.

The GRP\_WIDTH 64-bit, 128-bit, and 256-bit **SIGNALGRPs**, and trace data, have the limitation that the comparator is 128 bits and operates on the lower 128 signals, for example **SIGNALGRP<n>[127:0]**. Registers, such as SIGCOMPn and SIGMASKn, are therefore only 128 bits. The upper 128 bits of **SIGNALGRP<n>** must be used to trace data that does not require a trigger signal comparison. For example, an address, or control signals, can be used for trigger signal comparisons on **SIGNALGRP<n>[127:0]**, with the associated data being traced on **SIGNALGRP<n>[255:128]**.

The mask registers can also support *Trigger Conditions* that only use a single set of signals. They are:

### Debug signals only

This is achieved by masking out the eight *External Trigger Input Signals* and writing the *External Compare* register with zeros. This causes an always true condition.

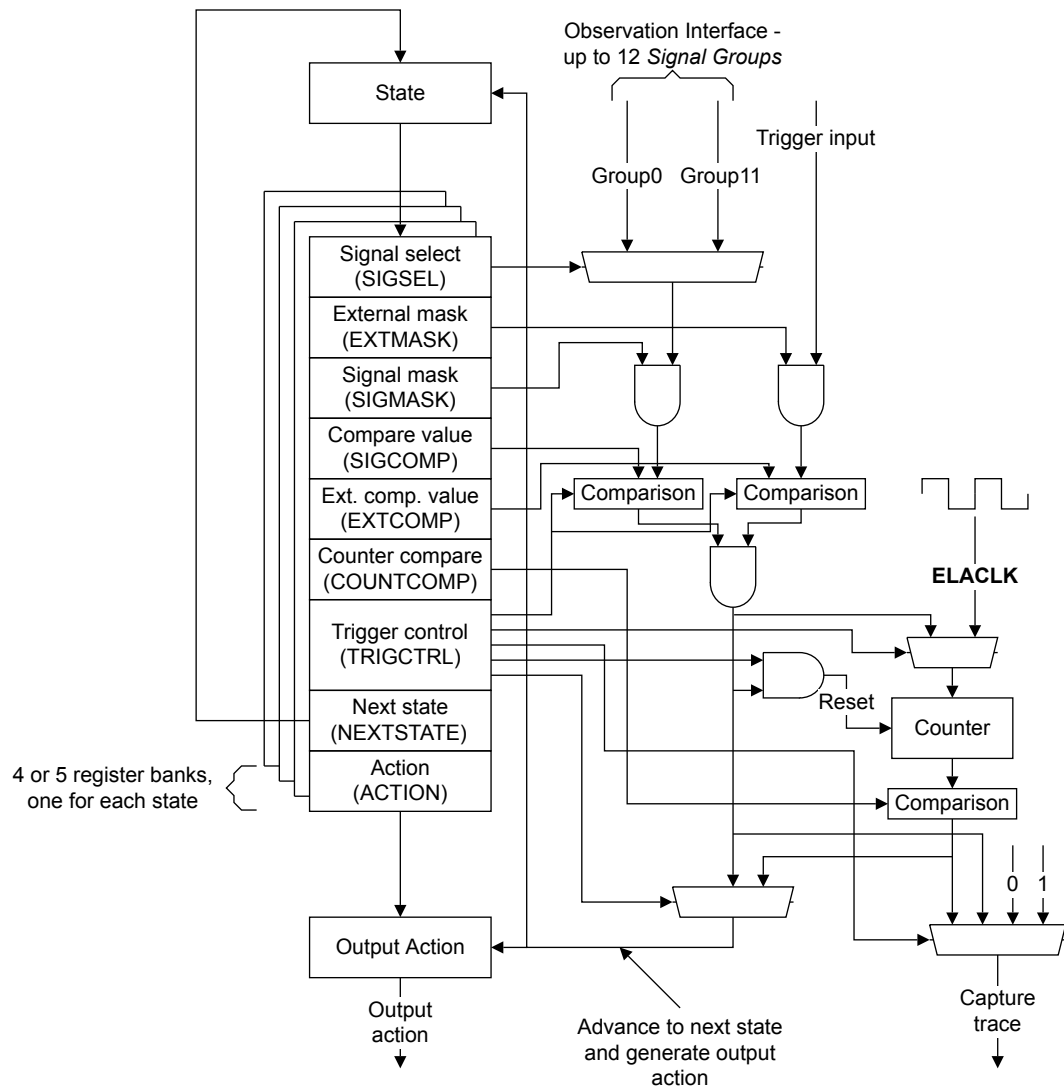
### External Trigger Input Signals only

This is achieved by masking out all the debug signals in the *Signal Group* and writing zeros into the *Signal Compare* register.

*Trigger Counter Comparisons* are made when the *Trigger State* counter counts from zero to the target value set in the *Counter Compare* register (COUNTCOMP<n>). A *Trigger Condition* can only be caused by either a *Trigger Signal Comparison* or a *Trigger Counter Comparison*.

*Trigger States* can be placed into loops by programming the NEXTSTATE<n> register to move to a previous *Trigger State*. *Trigger State* loops can be useful for trace filtering that is based on repeated *Trigger Signal Comparison* and *Trigger Counter Comparison* conditions. There is a *Trigger State* loop counter capability that can be enabled by programming TRIGCTRL.COUNTBRK = 1 and TRIGCTRL.COUNTCLR = 0. The loop counter can be used to break the loop and stop trace before the SRAM is full, or after a count of *Trigger Signal Comparison* events or a count of cycles. The *Trigger State* loop counter can also be used to control the number of ACTIONS by toggling ELA outputs **ELAOUTPUT** and **CTTRIGOUT**, which can be connected as interrupts or other CTI events.

The following figure shows the triggering mechanism within the ELA-500.



Note:  
Comparison represents a comparison function which can be one of:  
=, >, >=, !=, <, <=.

Figure 2-6 Triggering mechanism within the ELA-500

This section contains the following subsections:

- 2.5.1 Conditional trigger states on page 2-34.
- 2.5.2 Transaction ID capture on page 2-36.

### 2.5.1 Conditional trigger states

Conditional *Trigger States* adds an optional second trigger signal comparison to each *Trigger State* when the parameter COND\_TRIG = 1.

DEVID[19:16] = 1 indicates to software that the ELA-500 supports conditional *Trigger States*.

The following pseudocode examples show *Trigger State* comparisons using conditional *Trigger State* support, which adds the options that are highlighted in *italic text*:

Example 1: Both *Trigger State* conditions are programmed for trigger signal comparisons.

```
IF ((SIGNALGRP<n> < SIGCOMP0) && (TRIGCTRL0.COMPSEL == 0))
    drive ACTION0 and goto NEXTSTATE0
ELSE IF ((SIGNALGRP<n> > SIGCOMP0) && (TRIGCTRL0.ALTCOMPSEL == 0))
```

```
drive ALTACTION0 and goto ALTNEXTSTATE0
ELSE do nothing
```

Example 2: The first condition is programmed for trigger signal comparison, and the alternative condition is programmed for counter cycle count comparison. The event count can be programmed for the first IF condition.

```
IF ((SIGNALGRP<n> < SIGCOMP0) && (TRIGCTRL0.COMPSEL == 0))
    drive ACTION0 and goto NEXTSTATE0
ELSE IF ((counter == COUNTCOMP0) && (TRIGCTRL0.ALTCOMPSEL == 1) && (TRIGCTRL0.COUNTSRC == 0))
    drive ALTACTION0 and goto ALTNEXTSTATE0
ELSE do nothing
```

Example 3: The first condition is programmed for signal comparison, and the alternative condition is programmed for counter comparison using loop counter COUNTBRK.

```
IF ((SIGNALGRP<n> < SIGCOMP0) && (TRIGCTRL0.COMPSEL == 0))
    drive ACTION0 and goto NEXTSTATE0
ELSE IF ((counter == COUNTCOMP0) && (TRIGCTRL0.ALTCOMPSEL == 1) && (TRIGCTRL0.COUNTBRK == 1))
    IF((counter_match != 1) && (event_or_cycle_count_increment))
        goto ALTNEXTSTATE0
    ELSE IF (counter_match == 1)
        goto final_state
ELSE // COUNTBRK will default to the first IF condition branch
    goto NEXTSTATE0
```

Example 4: The first condition is programmed for counter comparison, and the alternative condition is programmed for signal comparison. In this case, the first condition, the counter match, dominates when there is a simultaneous signal comparison match.

```
IF ((counter == COUNTCOMP0) && (TRIGCTRL0.ALTCOMPSEL == 1) && (TRIGCTRL0.COUNTSRC == 0))
    drive ACTION0 and goto NEXTSTATE0
ELSE IF ((SIGNALGRP<n> < SIGCOMP0) && (TRIGCTRL0.COMPSEL == 0))
    drive ALTACTION0 and goto ALTNEXTSTATE0
ELSE do nothing
```

Example 5: The first condition is programmed for counter comparison using loop counter COUNTBRK, and the alternative condition is programmed for signal comparison.

```
IF ((counter == COUNTCOMP0) && (TRIGCTRL0.COMPSEL == 1) && (TRIGCTRL0.COUNTBRK == 1))
    IF ((counter_match != 1) && event_or_cycle_count_increment)
        goto NEXTSTATE0
    ELSE IF (counter_match == 1)
        goto final_state
ELSE IF ((SIGNALGRP<n> < SIGCOMP0) && (TRIGCTRL0.ALTCOMPSEL == 0))
    drive ALTACTION0 and goto ALTNEXTSTATE0
ELSE // COUNTBRK will default to the first IF condition branch
    goto NEXTSTATE0
```

Example 6: Counting trigger signal comparisons as events does not need to use an alternative condition.

```
Trigger State 0:
Signal Comparison:
If (SIGNALGRP<n> == SIGCOMP0)
    trace SIGNALGRP

Counter Comparison: // use the counter to count Signal Comparison events
If (counter == COUNTCOMP0) && (TRIGCTRL0.COMPSEL == 1) && (TRIGCTRL0.COUNTSRC == 1) &&
(TRIGCTRL0.COMP == 3'b001)
    goto final_state
ELSE IF (SIGNALGRP<n> == SIGCOMP0)
    counter = counter + 1
ELSE do nothing
```

### Note

Filtered trace that is based on signal comparisons or counter comparisons is still independently controlled by TRIGCTRL<n>.TRACE.

If both the first IF and alternative ELSE IF trigger conditions match on the same clock cycle comparison, the first IF condition dominates and drives ACTION0 and goes to NEXTSTATE0.

TRIGCTRLn.COUNTBRK = 1 loops to the primary IF condition branch if the primary IF or alternative ELSE IF conditions do not match in the comparison clock cycle.

Two additional registers are used to provide support for the conditional *Trigger States*.

### Related references

[3.10.5 Alt Next State registers on page 3-62.](#)

[3.10.6 Alt Action registers on page 3-63.](#)

## 2.5.2 Transaction ID capture

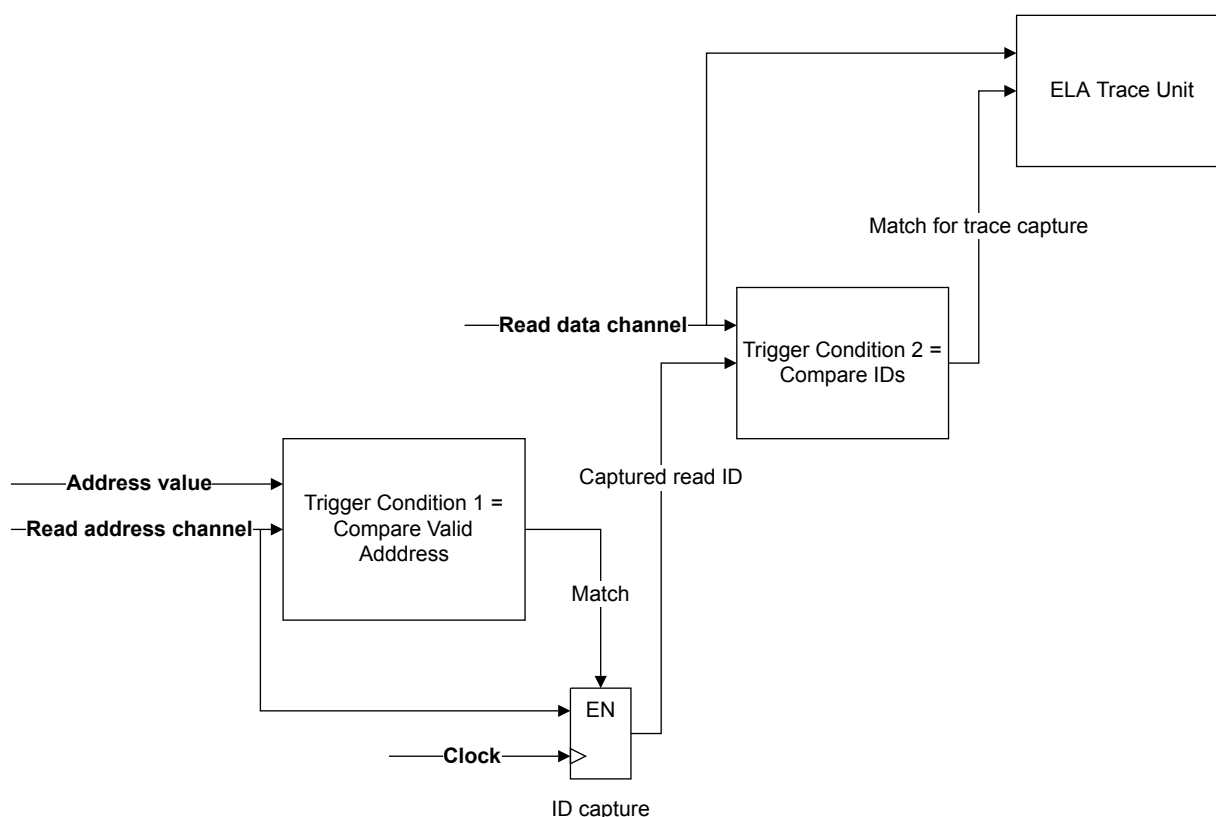
Captures a transaction ID when there is a trigger signal match of an address request in a *Trigger State*.

The ID bits are based on debug signal wiring to the LSB bits of the SIGNALGRP and the parameter ID\_CAPTURE\_SIZE which identifies the MSB of the ID, such that transaction\_ID[x:0] = SIGNALGRP<n>[x:0].

Stacked IDs can be wired to a SIGNALGRP such that {transaction\_ID1[y:x+1], transaction\_ID0[x:0]} = SIGNALGRP<n>[y:0].

SIGMASK<n> can be used to select the transaction ID in a subsequent trigger state.

The following figure illustrates a transaction ID capture from a read address channel on the first *Trigger State*, and use of the ID to compare with other debug signals connected to the SIGNALGRP<n> port in the second *Trigger State*.



**Figure 2-7 Transaction ID capture from read address channel**

The parameters ID\_CAPTURE\_GEN = 1 and COND\_TRIG = 1 must be set to enable generation of the ID capture feature. The ID\_CAPTURE\_SIZE parameter must be set to the number of bits in the ID tag.

If DEVID1 bits[24:20] = 0 then it indicates that the Transaction ID capture feature is disabled, otherwise DEVID1 bits[24:20] are set to ID\_CAPTURE\_SIZE. The parameter ID\_CAPTURE\_SIZE sets the size of the ID capture register which can be 2-30 bits.

#### **Related references**

[3.18.4 Device Configuration register on page 3-76.](#)

## 2.6 Authentication interface

The ELA-500 supports the authentication signals **DBGEN**, **NIDEN**, **SPIDEN**, and **SPNIDEN** through the Authentication interface.

When the ELA-500 is configured for Secure visibility, that is, the parameter `SECURE_MODE=1`:

- The ELA-500 *Trigger State* and trace operation are enabled when Secure non-invasive debug is enabled. When Secure non-invasive debug is disabled, the ELA-500 is stopped and does not move between *Trigger States*, assert any outputs, or capture any trace.

---

**Note**

- The ELA-500 does not operate if authentication signals are set to Non-secure debug mode.
  - The CoreSight Authentication rules require that Non-secure debug is enabled for Secure debug to be enabled.
- 

- The **STOPCLOCK** output is only asserted when Secure invasive debug is enabled.
- If Secure non-invasive debug becomes disabled dynamically, CTRL.RUN is internally gated, disabling the ELA-500. APB reads of CTRL.RUN do not indicate the state of the internally-gated run signal.

When the ELA-500 is not configured for Secure visibility, that is the parameter `SECURE_MODE=0`:

- The ELA-500 *Trigger State* and trace operation are enabled when Non-secure non-invasive debug is enabled. When Non-secure non-invasive debug is disabled, the ELA-500 is stopped and does not move between *Trigger States*, assert any outputs, or capture any trace.
- The **STOPCLOCK** output is only asserted when Non-secure invasive debug is enabled.
- If Non-secure non-invasive debug becomes disabled dynamically, CTRL.RUN is automatically set to 0, disabling the ELA-500.

---

**Note**

- ARM does not recommend the use of `SECURE_MODE=0`.
  - `SECURE_MODE = 1` is the default, and the setting that ARM recommends.
- 

For more information on the Authentication interface, including the permitted values of the Authentication signals, and the CoreSight debug states, see the *ARM® CoreSight™ Architecture Specification*.

## 2.7 Parameter summary

The functionality of the ELA-500 is determined by configurable parameters.

The following table shows the parameters that control the configuration of the ELA-500:

**Table 2-2 ELA-500 configuration parameters**

Parameter	Values	Default	Description
GRP_WIDTH	64, 128, or 256	128	64, 128, or 256 debug signals are allowed in a <i>Signal Group</i> .  When GRP_WIDTH = 64, the APB registers for the extended 128-bit SIGMASK and SICCOMP values are not available. When GRP_WIDTH = 256, signal comparisons are only made on the lower 128 bits of the <b>SIGNALGRP</b> .
RAM_ADDR_SIZE	2-25 inclusive	9	Number of address bits in the SRAM.  Number of entries in the SRAM = $2^{\text{RAM\_ADDR\_SIZE}}$  For example, when RAM_ADDR_SIZE is 9, the SRAM has 512 entries. ARM recommends at least 512 entries, where an entry is data payload plus the header byte.  When GRP_WIDTH = 64, each entry is 9 bytes. When GRP_WIDTH = 128, each entry is 17 bytes. When GRP_WIDTH = 256, each entry is 33 bytes.
TRACE_GEN	0 or 1	1	0: Trace unit not generated.  The APB trace registers are not available and the SRAM interface signals are tied to zero.
			1: Trace unit generated.
SECURE_MODE	0 or 1	1	0: Non-secure mode operation.  ————— <b>Note</b> ————— ARM does not recommend the use of Non-secure mode.  —————
			1: Secure mode operation.
COND_TRIG	0 or 1	0	0: Conditional <i>Trigger State</i> feature is disabled.
			1: Conditional <i>Trigger State</i> feature is enabled.
ID_CAPTURE_SIZE	2-30	10	Sets the number of bits used for the captured transaction ID.
ID_CAPTURE_GEN	0 or 1	0	0: Captured transaction ID is not generated.
			1: Captured transaction ID is generated.  ————— <b>Note</b> ————— When ID_CAPTURE_GEN = 1, you must also set COND_TRIG = 1.  —————
NUM_TRIG_STATES	4 or 5	4	Sets the number of <i>Trigger States</i> .
TRIGIN_EDGE	0 or 1	0	Used to determine the method by which <b>CTTRIGIN</b> and <b>EXTRIG</b> are detected.  When TRIGIN_EDGE = 0, <b>CTTRIGIN</b> and <b>EXTRIG</b> are sampled when the <i>Trigger State</i> signal comparison occurs. When TRIGIN_EDGE = 1, assertion of <b>CTTRIGIN</b> and <b>EXTRIG</b> are detected if CTRL.RUN = 1, and their assertions are latched until CTRL.RUN = 0.

# Chapter 3

## Programmers model

This chapter describes the programmers model.

---

**Note**

Any register bit position that is not listed in the description tables is reserved.

---

It contains the following sections:

- [3.1 Access permissions](#) on page 3-42.
- [3.2 Programming sequence](#) on page 3-43.
- [3.3 Control register summary](#) on page 3-44.
- [3.4 Control register descriptions](#) on page 3-45.
- [3.5 Current State register summary](#) on page 3-48.
- [3.6 Current State register descriptions](#) on page 3-49.
- [3.7 RAM register summary](#) on page 3-51.
- [3.8 RAM register descriptions](#) on page 3-52.
- [3.9 Trigger State register summary](#) on page 3-55.
- [3.10 Trigger State register descriptions](#) on page 3-58.
- [3.11 Integration Mode register summary](#) on page 3-67.
- [3.12 Integration Mode register descriptions](#) on page 3-68.
- [3.13 Software Lock register summary](#) on page 3-70.
- [3.14 Software Lock register descriptions](#) on page 3-71.
- [3.15 Authentication register summary](#) on page 3-72.
- [3.16 Authentication register descriptions](#) on page 3-73.
- [3.17 Device register summary](#) on page 3-74.
- [3.18 Device register descriptions](#) on page 3-75.
- [3.19 ID register summary](#) on page 3-78.



- [3.20 ID register descriptions on page 3-79.](#)

## 3.1 Access permissions

Individual registers and register groups have different availability, depending on whether the ELA-500 is running or not.

The following table lists the software access permissions by register group or individual register where appropriate.

**Table 3-1 Register access permissions**

Register or register group	Access when CTRL.RUN=1 <sup>a</sup>	Access when CTRL.RUN=0 <sup>a</sup>
<i>Logic Analyzer Control register on page 3-45</i>	Can be accessed by software.	Can be accessed by software.
<i>Current state registers on page 3-48</i>		
<i>Software lock registers on page 3-70</i>		
<i>Authentication registers on page 3-72</i>		
<i>Device registers on page 3-74</i>		
<i>ID registers on page 3-78</i>		
<i>RAM registers on page 3-51</i>	Must not be accessed by software.	
<i>Integration mode registers on page 3-67</i>		
<i>Trigger state registers on page 3-55</i>	Can be read by software. Must not be written.	
<i>Timestamp Control register on page 3-45</i>		
<i>Pre-trigger Action register on page 3-46</i>		

<sup>a</sup> Access means reads or writes, according to the Type field in the relevant register summary table.

## 3.2 Programming sequence

Programming the ELA-500 requires you to perform several steps.

---

### Note

---

- In the following sequence of steps, a lowercase  $\langle n \rangle$ , where  $n = 0$  to 4, denotes one of the five *Trigger States*.
  - The order of steps 2 to 8 is not important.
- 

### Procedure

1. Set CTRL.RUN = 0 to stop the logic analyzer and allow the *Trigger State* and conditions to be programmed.
2. Select the bus to be used for comparisons for each *Trigger State* by writing to the appropriate SIGSEL $\langle n \rangle$  register.
3. Program the mask and compare registers for each *Trigger State* by writing to SIGMASK $\langle n \rangle$  and SIGCOMP $\langle n \rangle$ , respectively.
4. Program the *Output Action* for each *Trigger State* by writing to the ACTION $\langle n \rangle$  registers. Program an initial *Output Action* by writing to the PTACTION register.  
*Trigger State* sequences and actions result in either a level or a pulse on the logic analyzer output signals, depending on the value in the *Output Action* registers for subsequent *Trigger States*.
5. Program the next *Trigger State* sequence in the NEXTSTATE $\langle n \rangle$  registers.  
*Trigger State* 0 is the first enabled state after reset.
6. If the counters are used for a *Trigger State*, then write the compare value for each *Trigger State* counter to COUNTCOMP $\langle n \rangle$ .
7. Program the Trigger Control register to select the comparison type and counter options for each *Trigger State* by writing to TRIGCTRL $\langle n \rangle$ .
8. Write to the RWAR to set the first RAM address to be written and to clear the WRAP bit.
9. Set CTRL.RUN = 1 to enable the ELA-500.

The Current *Trigger State*, Counter, and Actions registers can be read when CTRL.RUN = 0 or 1.

### Related references

[Chapter 3 Programmers model on page 3-40.](#)

### 3.3 Control register summary

This section gives a summary of the ELA-500 Control registers.

The following table shows the Control registers in offset order from the base address of the ELA-500.

**Table 3-2 Control registers summary**

Offset	Name	Type	Reset	Description
0x000	CTRL	RW	0x00000000	<a href="#">3.4.1 Logic Analyzer Control register on page 3-45</a>
0x004	TIMECTRL	RW	0x00000000 <sup>b</sup>	<a href="#">3.4.2 Timestamp Control register on page 3-45</a>
0x008	TSSR	RW	0x00000000	<a href="#">3.4.3 Trigger State Select Register on page 3-46</a>
0x010	PTACTION	RW	0x00000000	<a href="#">3.4.4 Pre-trigger Action register on page 3-46</a>

<sup>b</sup> Must be initialized by software before writing CTRL.RUN=1.

## 3.4 Control register descriptions

This section describes the ELA-500 Control registers.

[Table 3-2 Control registers summary on page 3-44](#) provides cross-references to individual registers.

This section contains the following subsections:

- [3.4.1 Logic Analyzer Control register on page 3-45.](#)
- [3.4.2 Timestamp Control register on page 3-45.](#)
- [3.4.3 Trigger State Select Register on page 3-46.](#)
- [3.4.4 Pre-trigger Action register on page 3-46.](#)

### 3.4.1 Logic Analyzer Control register

The ELA-500 Logic Analyzer Control register enables and disables the ELA-500.

The CTRL register characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.3 Control register summary on page 3-44.</a>

The following table shows the bit assignments.

**Table 3-3 CTRL register bit assignments**

Bits	Name	Function
[0]	RUN	Run control.
		0 ELA-500 disabled. Register programming permitted.
		1 ELA-500 enabled.

### 3.4.2 Timestamp Control register

The ELA-500 Timestamp Control register enables insertion of timestamps in trace, programming of the timestamp request interval, and determination of which two bits from the 16 trace counter bits are written to the upper two bits of the trace header byte.

The TIMECTRL register characteristics are:

<b>Usage constraints</b>	Writing when CTRL.RUN = 1 results in improper operation.
<b>Configurations</b>	Available when TRACE_GEN = 1.
<b>Attributes</b>	See <a href="#">3.3 Control register summary on page 3-44.</a>

The following table shows the bit assignments.

**Table 3-4 TIMECTRL register bit assignments**

Bits	Name	Function
[16]	TSEN	Timestamp Enable.
[15:12]	TSINT	Timestamp Interval.  When Timestamps are enabled, TSINT specifies the bit number of the 16-bit trace counter that causes a timestamp packet to be requested. The trace counter runs from <b>ELACLK</b> . When the specified bit changes, a timestamp packet is requested to be inserted into the trace SRAM when there is an <b>ELACLK</b> cycle during which trace data is not being captured. The ELA-500 does not insert back-to-back timestamps in the SRAM, even when TSINT causes multiple requests to be made.  When TSINT = 0, a timestamp is written when ACTION.TRACE disables trace. Looping <i>Trigger States</i> enable and then disable trace, causing timestamp writes. A timestamp is always written when CTRL.RUN is cleared and the previous trace write contained a data payload.
[11:8]	Reserved	-
[7:4]	TCSEL1	Trace Counter 1 select.  Selects the bit number of the 16-bit trace counter that is presented as Trace Counter[1] in the SRAM header byte.
[3:0]	TCSEL0	Trace Counter 0 select.  Selects the bit number of the 16-bit trace counter that is presented as Trace Counter[0] in the SRAM header byte.

#### Related references

[2.4.4 Trace SRAM format on page 2-30.](#)

### 3.4.3 Trigger State Select Register

The Trigger State Select Register enables and disables independent trace for *Trigger State 4*.

The TSSR characteristics are:

**Usage constraints** No usage constraints.

**Configurations** Only available when NUM\_TRIG\_STATES = 5.

**Attributes** See [3.3 Control register summary on page 3-44](#).

The following table shows the bit assignments.

**Table 3-5 CTRL register bit assignments**

Bits	Name	Function
[7:0]	ALTTS	Each bit identifies the trigger state that enables independent trace. Only trigger state 4 supports independent trace.  ALTTS[4]=0 <i>Trigger State 4 independent trace disabled.</i> ALTTS[4]=1 <i>Trigger State 4 independent trace enabled.</i>  All other bits read zero.

### 3.4.4 Pre-trigger Action register

PTACTION sets a level on the Action outputs immediately after CTRL.RUN is set, and before the first *Trigger Condition* has been met.

The PTACTION register characteristics are:

**Usage constraints** Writing when CTRL.RUN = 1 results in improper operation.

**Configurations** Available in all configurations.

**Attributes** See [3.3 Control register summary](#) on page 3-44.

The following table shows the bit assignments.

**Table 3-6 PTACTION register bit assignments**

Bits	Name	Function
[7:4]	ELAOUTPUT	Sets the value to drive on <b>ELAOUTPUT</b> [3:0].
[3]	TRACE	Enables trace.
[2]	STOPCLOCK	Sets the level to drive on <b>STOPCLOCK</b> .
[1:0]	CTTRIGOUT	Sets the value to drive on <b>CTTRIGOUT</b> [1:0].

## 3.5 Current State register summary

This section gives a summary of the ELA-500 Current State registers.

The following table shows the Current State registers in offset order from the base address of the ELA-500.

**Table 3-7 Current State registers summary**

Offset	Name	Type	Reset	Description
0x020	CTSR	RO	0b0001	<a href="#">3.6.1 Current Trigger State Register on page 3-49</a>
0x024	CCVR	RO	0x00000000	<a href="#">3.6.2 Current Counter Value Register on page 3-49</a>
0x028	CAVR	RO	0x00	<a href="#">3.6.3 Current Action Value Register on page 3-50</a>
0x02C	RDCAPTID	RO	-	<a href="#">3.6.4 Read Captured Transaction ID register on page 3-50</a>



## 3.6 Current State register descriptions

This section describes the ELA-500 Current State registers.

[Table 3-7 Current State registers summary on page 3-48](#) provides cross-references to individual registers.

This section contains the following subsections:

- [3.6.1 Current Trigger State Register on page 3-49.](#)
- [3.6.2 Current Counter Value Register on page 3-49.](#)
- [3.6.3 Current Action Value Register on page 3-50.](#)
- [3.6.4 Read Captured Transaction ID register on page 3-50.](#)

### 3.6.1 Current Trigger State Register

The Current Trigger State Register takes a snapshot of the current *Trigger State*. When the CTSR is read, the current values of the *Current Counter Value Register* (CCVR) and *Current Action Value Register* (CAVR) are also captured.

The CTSR characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.5 Current State register summary on page 3-48.</a>

The following table shows the bit assignments.

**Table 3-8 CTSR bit assignments**

Bits	Name	Function
[31]	FINALSTATE	<p>0 ELA-500 is still tracing.</p> <p>1 Indicates that the ELA-500 has stopped advancing <i>Trigger States</i> and stopped trace.</p> <p>FINALSTATE can be set by TRIGCTRL&lt;n&gt;.COUNTBRK reaching the final loop count, or by programming NEXTSTATE&lt;n&gt; or ALTNEXTSTATE&lt;n&gt; to zero.</p>
[30:NUM_TRIG_STATES]	Reserved	-
[NUM_TRIG_STATES-1:0]	CTSR	<p>Reads current <i>Trigger State</i>. This is a one-hot encoded field.</p> <p>When CTRL.RUN:</p> <p>0 RAZ.</p> <p>1 Returns current <i>Trigger State</i>.</p> <p>If FINALSTATE is 1, then the CTSR field gives the <i>Trigger State</i> when FINALSTATE became 1.</p>

### 3.6.2 Current Counter Value Register

The Current Counter Value Register returns the counter value that was captured when the *Current Trigger State Register* (CTSR) was read.

The CCVR characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.5 Current State register summary on page 3-48.</a>

The following table shows the bit assignments.

**Table 3-9 CCVR bit assignments**

Bits	Name	Function
[31:0]	CCVR	Returns the counter value when the CTSR was last read. If the CTSR has never been read, then the value in the CCVR is undefined.

### 3.6.3 Current Action Value Register

The Current Action Value Register returns the Action value that was captured when the *Current Trigger State Register* (CTSR) was read.

The CAVR characteristics are:

**Usage constraints** No usage constraints.

**Configurations** Available in all configurations.

**Attributes** See [3.5 Current State register summary on page 3-48](#).

The following table shows the bit assignments.

**Table 3-10 CAVR bit assignments**

Bits	Name	Function
[7:4]	ELAOUTPUT	Value driven on <b>ELAOUTPUT</b> [3:0].
[3]	TRACE	Trace active.  0b0 Trace is not active. 0b1 Trace is active.
[2]	STOPCLOCK	Level driven on <b>STOPCLOCK</b> .  0b0 0 Driven on <b>STOPCLOCK</b> . 0b1 1 Driven on <b>STOPCLOCK</b> .
[1:0]	CTTRIGOUT	Value driven on <b>CTTRIGOUT</b> [1:0].

### 3.6.4 Read Captured Transaction ID register

The RDCAPTID register captures a transaction ID on the trigger signal match of an address request in a *Trigger State*. The ID can then be used by a subsequent *Trigger State* to trace the response to the original request.

The RDCAPTID register characteristics are:

**Usage constraints** Can only be read when CTRL.RUN = 0 and ID\_CAPTURE\_GEN = 1.

**Configurations** Available in all configurations.

**Attributes** See [3.5 Current State register summary on page 3-48](#).

The following table shows the bit assignments.

**Table 3-11 RDCAPTID bit assignments**

Bits	Name	Function
[ID_CAPTURE_SIZE-1:0]	RDCAPTID	Returns the captured transaction ID.

## 3.7 RAM register summary

This section gives a summary of the ELA-500 RAM registers.

The following table shows the RAM registers in offset order from the base address of the ELA-500.

**Table 3-12 RAM registers summary**

Offset	Name	Type	Reset	Description
0x040	RRAR	RW	-	<a href="#">3.8.1 RAM Read Address Register on page 3-52</a>
0x044	RRDR	RO	-	<a href="#">3.8.2 RAM Read Data Register on page 3-52</a>
0x048	RWAR	RW	-	<a href="#">3.8.3 RAM Write Address Register on page 3-53</a>
0x04C	RWDR	WO	-	<a href="#">3.8.4 RAM Write Data Register on page 3-53</a>

## 3.8 RAM register descriptions

This section describes the ELA-500 RAM registers.

[Table 3-12 RAM registers summary on page 3-51](#) provides cross-references to individual registers.

This section contains the following subsections:

- [3.8.1 RAM Read Address Register on page 3-52.](#)
- [3.8.2 RAM Read Data Register on page 3-52.](#)
- [3.8.3 RAM Write Address Register on page 3-53.](#)
- [3.8.4 RAM Write Data Register on page 3-53.](#)

### 3.8.1 RAM Read Address Register

The RAM Read Address Register is used to select the address that is read from the trace SRAM into a holding register.

The RRAR characteristics are:

- Usage constraints** No access when CTRL.RUN = 1.
- Configurations** Only available in configurations with TRACE\_GEN = 1.
- Attributes** See [3.7 RAM register summary on page 3-51](#).

The following table shows the bit assignments.

**Table 3-13 RRAR bit assignments**

Bits	Name	Function
[RAM_ADDR_SIZE-1:0]	RRA	<p>RAM Read Address.</p> <p>Writes to the RRA cause the trace SRAM data at that address to be transferred into the holding register.</p> <p>After the SRAM read data is transferred to the holding register, RRA increments by one. This prepares the RRA address for sequential RRDR reads.</p> <p>The RRA automatically increments after APB reads from the RRDR have read the contents of the holding register. An RRDR read of the last data in the holding register initiates a read to SRAM at the address pointed to by the RRA. The holding register is filled with the data at this address, then the RRA increments.</p>

#### Related references

[3.8.2 RAM Read Data Register on page 3-52.](#)

### 3.8.2 RAM Read Data Register

The RAM Read Data Register is a read-only register that reads data from the SRAM read holding register.

The RRDR characteristics are:

- Usage constraints** No access when CTRL.RUN = 1.
- Configurations** Only available in configurations with TRACE\_GEN = 1.
- Attributes** See [3.7 RAM register summary on page 3-51](#).

The following table shows the bit assignments.

**Table 3-14 RRDR bit assignments**

Bits	Name	Function
[31:0]	RRD	<p>Reads SRAM data from the holding register.</p> <p>Reads from the RRD return the SRAM data from the holding register. The first read of the RRD after an RRAR update returns the trace data header byte value, zero-extended to 32 bits. Subsequent reads of the RRD return 32-bit chunks of the trace data payload, starting with the least significant word, until all the payload data has been read, that is, two words if GRP_WIDTH = 64, four words if GRP_WIDTH = 128, and eight words if GRP_WIDTH = 256.</p> <p>When the final 32 bits of the payload have been read, the RRA is incremented automatically, and the next word of SRAM data is copied into the holding register. This enables the SRAM data content to be read out efficiently.</p> <p>The RRA wraps to address zero if it is incremented beyond the maximum depth of the SRAM.</p>

### 3.8.3 RAM Write Address Register

The RAM Write Address Register is used to select the SRAM address that the data from the write holding register is written to.

The RWAR characteristics are:

- Usage constraints** No access when CTRL.RUN = 1.
- Configurations** Only available in configurations with TRACE\_GEN = 1.
- Attributes** See [3.7 RAM register summary on page 3-51](#).

The following table shows the bit assignments.

**Table 3-15 RWAR bit assignments**

Bits	Name	Function
[31]	WRAP	The WRAP bit is set when the RAM Write Address is incremented beyond 2 <sup>RAM_ADDR_SIZE</sup> while the ELA-500 is capturing trace data. The WRAP bit is not set by writes to the RWDR that cause the RAM Write Address to roll over. Software must clear the WRAP bit when writing to the RWAR.
[RAM_ADDR_SIZE-1:0]	RWA	<p>RAM Write Address.</p> <p>Writes to the RWA set the SRAM address for data that is then written through the RWDR.</p> <p>Reads from the RWA return the address of the SRAM location that is to be written next, either by writes to the RWDR, or by the trace unit.</p> <p>When trace is stopped, the RWA contains the address of the last SRAM location that was written plus one. If the RAM Write Address was incremented beyond the depth of the RAM while the ELA-500 was capturing trace data, the WRAP bit is set.</p> <p>The RWAR is automatically incremented by APB writes to the SRAM through the RWDR.</p>

#### Related references

[3.8.4 RAM Write Data Register on page 3-53](#).

### 3.8.4 RAM Write Data Register

The RAM Write Data Register is a write-only register that writes data to the SRAM write holding register.

The RWDR characteristics are:

- Usage constraints** No access when CTRL.RUN = 1.
- Configurations** Only available in configurations with TRACE\_GEN = 1.

**Attributes** See [3.7 RAM register summary on page 3-51](#).

The following table shows the bit assignments.

**Table 3-16 RWDR bit assignments**

Bits	Name	Function
[31:0]	RWDR	<p>Writes data to the write holding register and initiates an SRAM write when the write holding register is full.</p> <p>Writes to the RWD update the internal write holding register.</p> <p>The first write to the RWD sets the header byte value from the least significant byte written. Subsequent writes to the RWD set 32-bit chunks of the payload, starting with the least significant chunk. When the final 32 bits of the payload have been written, the content of the holding register is copied into the SRAM and the RWA is incremented automatically.</p>

### 3.9 Trigger State register summary

This section gives a summary of the ELA-500 *Trigger State* registers.

The following table shows the trigger state registers in offset order from the base address of the ELA-500.

**Note**

All the *Trigger State* registers must be initialized by software before writing CTRL.RUN=1.

**Table 3-17 Trigger state registers summary**

Offset	Name	Type	Reset	Description
<i>Trigger State 0 registers</i>				
0x100	SIGSEL0	RW	-	<a href="#">3.10.1 Signal Select registers on page 3-58</a>
0x104	TRIGCTRL0	RW	-	<a href="#">3.10.2 Trigger Control registers on page 3-59</a>
0x108	NEXTSTATE0	RW	-	<a href="#">3.10.3 Next State registers on page 3-61</a>
0x10C	ACTION0	RW	0x00	<a href="#">3.10.4 Action registers on page 3-61</a>
0x110	ALTNEXTSTATE0	RW	-	<a href="#">3.10.5 Alt Next State registers on page 3-62</a>
0x114	ALTACTION0	RW	0x00	<a href="#">3.10.6 Alt Action registers on page 3-63</a>
0x120	COUNTCOMP0	RW	-	<a href="#">3.10.7 Counter Compare registers on page 3-64</a>
0x130	EXTMASK0	RW	-	<a href="#">3.10.8 External Mask registers on page 3-64</a>
0x134	EXTCOMP0	RW	-	<a href="#">3.10.9 External Compare registers on page 3-64</a>
0x140	SIGMASK0[31:0]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x144	SIGMASK0[63:32]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x148	SIGMASK0[95:64]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x14C	SIGMASK0[127:96]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x180	SIGCOMP0[31:0]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x184	SIGCOMP0[63:32]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x188	SIGCOMP0[95:64]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x18C	SIGCOMP0[127:96]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
<i>Trigger State 1 registers</i>				
0x200	SIGSEL1	RW	-	<a href="#">3.10.1 Signal Select registers on page 3-58</a>
0x204	TRIGCTRL1	RW	-	<a href="#">3.10.2 Trigger Control registers on page 3-59</a>
0x208	NEXTSTATE1	RW	-	<a href="#">3.10.3 Next State registers on page 3-61</a>
0x20C	ACTION1	RW	0x00	<a href="#">3.10.4 Action registers on page 3-61</a>
0x210	ALTNEXTSTATE1	RW	-	<a href="#">3.10.5 Alt Next State registers on page 3-62</a>
0x214	ALTACTION1	RW	0x00	<a href="#">3.10.6 Alt Action registers on page 3-63</a>
0x220	COUNTCOMP1	RW	-	<a href="#">3.10.7 Counter Compare registers on page 3-64</a>
0x230	EXTMASK1	RW	-	<a href="#">3.10.8 External Mask registers on page 3-64</a>

**Table 3-17 Trigger state registers summary (continued)**

Offset	Name	Type	Reset	Description
0x234	EXTCOMP1	RW	-	<a href="#">3.10.9 External Compare registers on page 3-64</a>
0x240	SIGMASK1[31:0]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x244	SIGMASK1[63:32]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x248	SIGMASK1[95:64]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x24C	SIGMASK1[127:96]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x280	SIGCOMP1[31:0]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x284	SIGCOMP1[63:32]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x288	SIGCOMP1[95:64]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x28C	SIGCOMP1[127:96]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
<i>Trigger State 2 registers</i>				
0x300	SIGSEL2	RW	-	<a href="#">3.10.1 Signal Select registers on page 3-58</a>
0x304	TRIGCTRL2	RW	-	<a href="#">3.10.2 Trigger Control registers on page 3-59</a>
0x308	NEXTSTATE2	RW	-	<a href="#">3.10.3 Next State registers on page 3-61</a>
0x30C	ACTION2	RW	0x00	<a href="#">3.10.4 Action registers on page 3-61</a>
0x310	ALTNEXTSTATE2	RW	-	<a href="#">3.10.5 Alt Next State registers on page 3-62</a>
0x314	ALTACTION2	RW	0x00	<a href="#">3.10.6 Alt Action registers on page 3-63</a>
0x320	COUNTCOMP2	RW	-	<a href="#">3.10.7 Counter Compare registers on page 3-64</a>
0x330	EXTMASK2	RW	-	<a href="#">3.10.8 External Mask registers on page 3-64</a>
0x334	EXTCOMP2	RW	-	<a href="#">3.10.9 External Compare registers on page 3-64</a>
0x340	SIGMASK2[31:0]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x344	SIGMASK2[63:32]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x348	SIGMASK2[95:64]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x34C	SIGMASK2[127:96]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x380	SIGCOMP2[31:0]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x384	SIGCOMP2[63:32]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x388	SIGCOMP2[95:64]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x38C	SIGCOMP2[127:96]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
<i>Trigger State 3 registers</i>				
0x400	SIGSEL3	RW	-	<a href="#">3.10.1 Signal Select registers on page 3-58</a>
0x404	TRIGCTRL3	RW	-	<a href="#">3.10.2 Trigger Control registers on page 3-59</a>
0x408	NEXTSTATE3	RW	-	<a href="#">3.10.3 Next State registers on page 3-61</a>
0x40C	ACTION3	RW	0x00	<a href="#">3.10.4 Action registers on page 3-61</a>
0x410	ALTNEXTSTATE3	RW	-	<a href="#">3.10.5 Alt Next State registers on page 3-62</a>
0x414	ALTACTION3	RW	0x00	<a href="#">3.10.6 Alt Action registers on page 3-63</a>
0x420	COUNTCOMP3	RW	-	<a href="#">3.10.7 Counter Compare registers on page 3-64</a>



**Table 3-17 Trigger state registers summary (continued)**

Offset	Name	Type	Reset	Description
0x430	EXTMASK3	RW	-	<a href="#">3.10.8 External Mask registers on page 3-64</a>
0x434	EXTCOMP3	RW	-	<a href="#">3.10.9 External Compare registers on page 3-64</a>
0x440	SIGMASK3[31:0]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x444	SIGMASK3[63:32]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x448	SIGMASK3[95:64]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x44C	SIGMASK3[127:96]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x480	SIGCOMP3[31:0]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x484	SIGCOMP3[63:32]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x488	SIGCOMP3[95:64]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x48C	SIGCOMP3[127:96]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
<i>Trigger State 4 registers</i>				
0x500	SIGSEL4	RW	-	<a href="#">3.10.1 Signal Select registers on page 3-58</a>
0x504	TRIGCTRL4	RW	-	<a href="#">3.10.2 Trigger Control registers on page 3-59</a>
0x508	NEXTSTATE4	RW	-	<a href="#">3.10.3 Next State registers on page 3-61</a>
0x50C	ACTION4	RW	0x00	<a href="#">3.10.4 Action registers on page 3-61</a>
0x510	ALTNEXTSTATE4	RW	-	<a href="#">3.10.5 Alt Next State registers on page 3-62</a>
0x514	ALTACTION4	RW	0x00	<a href="#">3.10.6 Alt Action registers on page 3-63</a>
0x520	COUNTCOMP4	RW	-	<a href="#">3.10.7 Counter Compare registers on page 3-64</a>
0x530	EXTMASK4	RW	-	<a href="#">3.10.8 External Mask registers on page 3-64</a>
0x534	EXTCOMP4	RW	-	<a href="#">3.10.9 External Compare registers on page 3-64</a>
0x540	SIGMASK4[31:0]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x544	SIGMASK4[63:32]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x548	SIGMASK4[95:64]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x54C	SIGMASK4[127:96]	RW	-	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>
0x580	SIGCOMP4[31:0]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x584	SIGCOMP4[63:32]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x588	SIGCOMP4[95:64]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>
0x58C	SIGCOMP4[127:96]	RW	-	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>

## 3.10 Trigger State register descriptions

This section describes the ELA-500 *Trigger State* registers.

[Table 3-17 Trigger state registers summary on page 3-55](#) provides cross-references to individual registers.

---

### Note

---

In the following register descriptions, a lowercase *<n>*, where *n* = 0 to 4, denotes one of the five *Trigger States*. For example, the SIGSEL2 register selects which input bus is used when the ELA-500 is in *Trigger State 2*.

---

This section contains the following subsections:

- [3.10.1 Signal Select registers on page 3-58.](#)
- [3.10.2 Trigger Control registers on page 3-59.](#)
- [3.10.3 Next State registers on page 3-61.](#)
- [3.10.4 Action registers on page 3-61.](#)
- [3.10.5 Alt Next State registers on page 3-62.](#)
- [3.10.6 Alt Action registers on page 3-63.](#)
- [3.10.7 Counter Compare registers on page 3-64.](#)
- [3.10.8 External Mask registers on page 3-64.](#)
- [3.10.9 External Compare registers on page 3-64.](#)
- [3.10.10 Signal Mask registers on page 3-65.](#)
- [3.10.11 Signal Compare registers on page 3-66.](#)

### 3.10.1 Signal Select registers

The Signal Select registers control the selection of the debug signal bus that is masked using the Signal Mask, and compared to the Signal Compare registers for all five *Trigger States*.

The SIGSEL<n> register characteristics are:

<b>Usage constraints</b>	Writing when CTRL.RUN = 1 results in improper operation.
<b>Configurations</b>	Available in all configurations. SIGSEL4 is only available when NUM_TRIG_STATES = 5.
<b>Attributes</b>	See <a href="#">3.9 Trigger State register summary on page 3-55</a> .

The following table shows the bit assignments.

**Table 3-18 SIGSEL<n> register bit assignments**

Bits	Name	Function
[31:12]	Reserved	-
[11:0]	SIGSEL<n>	<p>Selects <i>Signal Group</i>.</p> <p>0x1                Selects <i>Signal Group</i> 0.</p> <p>0x2                Selects <i>Signal Group</i> 1.</p> <p>0x4                Selects <i>Signal Group</i> 2.</p> <p>0x8                Selects <i>Signal Group</i> 3.</p> <p>0x10               Selects <i>Signal Group</i> 4.</p> <p>0x20               Selects <i>Signal Group</i> 5.</p> <p>0x40               Selects <i>Signal Group</i> 6.</p> <p>0x80               Selects <i>Signal Group</i> 7.</p> <p>0x100              Selects <i>Signal Group</i> 8.</p> <p>0x200              Selects <i>Signal Group</i> 9.</p> <p>0x400              Selects <i>Signal Group</i> 10.</p> <p>0x800              Selects <i>Signal Group</i> 11.</p>

### 3.10.2 Trigger Control registers

The Trigger Control registers are used to select the comparison type for each *Trigger State*. The comparisons are between the input *Signal Group* that is masked by the Signal Mask register, and the Signal Compare Registers.

The TRIGCTRL<n> register characteristics are:

**Usage constraints**      Writing when CTRL.RUN = 1 results in improper operation.

**Configurations**        Available in all configurations. TRIGCTRL4 is only available when  
NUM\_TRIG\_STATES = 5.

**Attributes**              See [3.9 Trigger State register summary on page 3-55](#).

The following table shows the bit assignments.

**Table 3-19 TRIGCTRL<n> register bit assignments**

Bits	Name	Function
[15]	ALTCOMPSEL	<p>Selects the alternative comparison mode:</p> <p>0b0                <i>Trigger Signal Alternative Comparisons</i> selected.</p> <p>0b1                <i>Trigger Counter Alternative Comparisons</i> selected.</p>
[14:12]	ALTCOMP	<p><i>Trigger Signal Alternative Comparison</i> type select:</p> <p>0b000              <i>Trigger Signal Alternative Comparisons</i> disabled.</p> <p>0b001              Alternative compare type is <i>equal</i> (==).</p> <p>0b010              Alternative compare type is <i>greater than</i> (&gt;).</p> <p>0b011              Alternative compare type is <i>greater than or equal</i> (&gt;=).</p> <p>0b101              Alternative compare type is <i>not equal</i> (!=).</p> <p>0b110              Alternative compare type is <i>less than</i> (&lt;).</p> <p>0b111              Alternative compare type is <i>less than or equal</i> (&lt;=).</p>

**Table 3-19 TRIGCTRL<n> register bit assignments (continued)**

Bits	Name	Function
[11:10]	CAPTID	<p><b>0b00</b> Disable use of the captured ID for signal comparisons.</p> <p><b>0b01</b> Capture ID when trigger signal condition matches.</p> <p>The ID is captured, from <b>SIGNALGRP&lt;n&gt;</b>[ID_CAPTURE_SIZE-1:0].</p> <p><b>0b10</b> Use the captured ID instead of the target value in <b>SIGCOMP&lt;n&gt;</b>[ID_CAPTURE_SIZE-1:0] for comparison of <b>SIGNALGRP&lt;n&gt;</b>[ID_CAPTURE_SIZE-1:0].</p> <p><b>0b11</b> Use the captured ID instead of the <b>SIGNALGRP&lt;n&gt;</b>[ID_CAPTURE_SIZE-1:0] for a comparison against <b>SIGCOMP&lt;n&gt;</b>[ID_CAPTURE_SIZE-1:0].</p>
[9]	COUNTBRK	<p>Loop counter break.</p> <p>The loop counter break uses the <i>Trigger State</i> counter to break loops between <i>Trigger States</i> after a <i>Trigger Counter Comparison</i>. When the counter comparison matches, the <i>Trigger State</i> goes into a final state, which stops trace writes and leaves the output actions at the previous <i>Trigger State</i> ACTION value.</p> <p><b>0b0</b> Normal operation.</p> <p><b>0b1</b> Break <i>Trigger State</i> loop: A counter comparison match causes a transition to the final state, otherwise go to the NEXTSTATE&lt;n&gt; <i>Trigger State</i> as the counter increments.</p>
[8]	COUNTCLR	<p>Counter clear.</p> <p><b>0b0</b> Do not clear the counter value when moving to a different NEXTSTATE&lt;n&gt;.</p> <p><b>0b1</b> Clear the counter value when moving to a different NEXTSTATE&lt;n&gt;.</p> <p>————— <b>Note</b> —————</p> <p>TRIGCTRL.WATCHRST must be <b>0b0</b> when using this feature.</p> <p>—————</p>
[7:6]	TRACE	<p>Trace capture control.</p> <p><b>0b00</b> Trace is captured when <i>Trigger Signal Comparison</i> succeeds.</p> <p><b>0b01</b> Trace is captured when <i>Trigger Counter Comparison</i> succeeds.</p> <p><b>0b10</b> Trace is captured every <b>ELACLK</b> cycle.</p> <p><b>0b11</b> Reserved.</p>
[5]	COUNTSRC	<p>Counter source select.</p> <p><b>0b0</b> Counter is incremented every <b>ELACLK</b> cycle.</p> <p><b>0b1</b> Counter is incremented when <i>Trigger Signal Comparison</i> matches.</p>
[4]	WATCHRST	<p>Counter reset.</p> <p><b>0b0</b> Do not reset the counter after a <i>Trigger Signal Comparison</i> match.</p> <p><b>0b1</b> Reset the counter after a <i>Trigger Signal Comparison</i> match.</p> <p>The counter acts like an activity watchdog timer, only allowing advancement to the next <i>Trigger State</i> when the <i>Trigger Counter Comparison</i> is reached. The counter is reset by a signal comparison.</p>

**Table 3-19 TRIGCTRL<n> register bit assignments (continued)**

Bits	Name	Function
[3]	COMPSEL	Comparison mode. Acts as both a counter enable and a select for the comparison mode.  0b0          Disable counters and select <i>Trigger Signal Comparison</i> mode. 0b1          Enable counters and select <i>Trigger Counter Comparison</i> mode.
[2:0]	COMP	<i>Trigger Signal Comparison</i> type select.  0b000 <i>Trigger Signal Comparisons</i> disabled. The enabled counters count clocks immediately after the <i>Trigger State</i> has been entered and generate a programmable <i>Output Action</i> and transition to the next <i>Trigger State</i> when the Counter Compare Register count is reached, that is when a <i>Trigger Counter Comparison</i> match occurs.  0b001      Compare type is <i>equal</i> (==). 0b010      Compare type is <i>greater than</i> (>). 0b011      Compare type is <i>greater than or equal</i> (>=). 0b101      Compare type is <i>not equal</i> (!=). 0b110      Compare type is <i>less than</i> (<). 0b111      Compare type is <i>less than or equal</i> (<=).

### 3.10.3 Next State registers

The Next State registers are zero-one-hot encoded registers that point to the next *Trigger State* that is entered after the *Trigger Condition* is met.

The NEXTSTATE<n> register characteristics are:

**Usage constraints**      Writing when CTRL.RUN = 1 results in improper operation.

**Configurations**          Available in all configurations. NEXTSTATE4 is only available when NUM\_TRIG\_STATES = 5.

**Attributes**              See [3.9 Trigger State register summary on page 3-55](#).

The following table shows the bit assignments.

**Table 3-20 NEXTSTATE<n> register bit assignments**

Bits	Name	Function
[NUM_TRIG_STATES-1:0]	NEXTSTATE<n>	Selects the next state to move to after the <i>Trigger Condition</i> has been met in the current state.  0x0          Do not change state. This is the final <i>Trigger State</i> . 0x1          Selects <i>Trigger State</i> 0. 0x2          Selects <i>Trigger State</i> 1. 0x4          Selects <i>Trigger State</i> 2. 0x8          Selects <i>Trigger State</i> 3. 0x10        Selects <i>Trigger State</i> 4, when NUM_TRIG_STATES = 5.

### 3.10.4 Action registers

The ACTION registers enable and disable trace, and control the level of the logic analyzer outputs on the ELAOUTPUT[3:0], STOPCLOCK, and CTTRIGOUT[1:0] pins.

The ACTION<n> register characteristics are:

<b>Usage constraints</b>	Writing when CTRL.RUN = 1 results in improper operation.
<b>Configurations</b>	Available in all configurations. ACTION4 is only available when NUM_TRIG_STATES = 5.
<b>Attributes</b>	See <a href="#">3.9 Trigger State register summary</a> on page 3-55.

The following table shows the bit assignments.

**Table 3-21 ACTION<n> register bit assignments**

Bits	Name	Function
[7:4]	ELAOUTPUT	Value to drive on <b>ELAOUTPUT[3:0]</b> .
[3]	TRACE	Trace active.  0b0 Trace is not active. 0b1 Trace is active.
[2]	STOPCLOCK	Level to drive on <b>STOPCLOCK</b> .  0b0 Drive 0 on <b>STOPCLOCK</b> . 0b1 Drive 1 on <b>STOPCLOCK</b> .
[1:0]	CTTRIGOUT	Value to drive on <b>CTTRIGOUT[1:0]</b> .

**Note**

ARM recommends the following **ELAOUTPUT** connections:

- **CTTRIGOUT[1:0]** - Connect to a *Cross Trigger Interface* (CTI) to enable the ELA-500 to trigger devices in the CoreSight system.
- **STOPCLOCK** - Connect to clocks unit to stop all clocks for serial scan dump.
- **ELAOUTPUT[3:0]** - Connect to any external device, for example a *Generic Interrupt Controller* (GIC) as an *Interrupt Request* (IRQ) input, an oscilloscope, or another ELA-500.

### 3.10.5 Alt Next State registers

The Alt Next State registers are zero-one-hot encoded registers that point to the next *Trigger State* that is entered after the conditional *Trigger Condition* is met.

The ALTNEXTSTATE<n> register characteristics are:

<b>Usage constraints</b>	Writing when CTRL.RUN = 1 results in improper operation.
<b>Configurations</b>	Available when COND_TRIG = 1. ALTNEXTSTATE4 is only available when NUM_TRIG_STATES = 5.
<b>Attributes</b>	See <a href="#">3.9 Trigger State register summary</a> on page 3-55.

The following table shows the bit assignments.

**Table 3-22 ALTNEXTSTATE<n> register bit assignments**

Bits	Name	Function
[NUM_TRIG_STATES-1:0]	ALTNEXTSTATE<n>	<p>Selects the next state to move to after the conditional <i>Trigger Condition</i> has been met in the current state.</p> <p>0x0 Do not change state. This is the final <i>Trigger State</i>.</p> <p>0x1 Selects <i>Trigger State</i> 0.</p> <p>0x2 Selects <i>Trigger State</i> 1.</p> <p>0x4 Selects <i>Trigger State</i> 2.</p> <p>0x8 Selects <i>Trigger State</i> 3.</p> <p>0x10 Selects <i>Trigger State</i> 4, when NUM_TRIG_STATES = 5.</p>

### 3.10.6 Alt Action registers

The Alt Action registers enable and disable trace and control the level of the logic analyzer outputs on the **ELAOUTPUT[3:0]**, **STOPCLOCK**, and **CTTRIGOUT[1:0]** pins.

The ALTACTION<n> register characteristics are:

**Usage constraints** Writing when CTRL.RUN = 1 results in improper operation.

**Configurations** Available when COND\_TRIG = 1. ALTACTION4 is only available when NUM\_TRIG\_STATES = 5.

**Attributes** See [3.9 Trigger State register summary on page 3-55](#).

The following table shows the bit assignments.

**Table 3-23 ALTACTION<n> register bit assignments**

Bits	Name	Function
[7:4]	ELAOUTPUT	Value to drive on <b>ELAOUTPUT[3:0]</b> .
[3]	TRACE	<p>Trace active.</p> <p>0b0 Trace is not active.</p> <p>0b1 Trace is active.</p>
[2]	STOPCLOCK	<p>Level to drive on <b>STOPCLOCK</b>.</p> <p>0b0 Drive 0 on <b>STOPCLOCK</b>.</p> <p>0b1 Drive 1 on <b>STOPCLOCK</b>.</p>
[1:0]	CTTRIGOUT	Value to drive on <b>CTTRIGOUT[1:0]</b> .

#### Note

ARM recommends the following **ELAOUTPUT** connections:

- **CTTRIGOUT[1:0]** - Connect to a *Cross Trigger Interface* (CTI) to enable the ELA-500 to trigger devices in the CoreSight system.
- **STOPCLOCK** - Connect to clocks unit to stop all clocks for serial scan dump.
- **ELAOUTPUT[3:0]** - Connect to any external device, for example a *Generic Interrupt Controller* (GIC) as an *Interrupt Request* (IRQ) input, an oscilloscope, or another ELA-500.

### 3.10.7 Counter Compare registers

The Counter Compare registers are used when *Trigger Counter Comparison* is selected in the appropriate TRIGCTRL<n> register, that is when TRIGCTRL<n>.COUNTEN = 1.

The COUNTCOMP<n> register characteristics are:

**Usage constraints** Writing when CTRL.RUN = 1 results in improper operation.

**Configurations** Available in all configurations. COUNTCOMP4 is only available when NUM\_TRIG\_STATES = 5.

**Attributes** See [3.9 Trigger State register summary on page 3-55](#).

The following table shows the bit assignments.

**Table 3-24 COUNTCOMP<n> register bit assignments**

Bits	Name	Function
[31:0]	COUNTCOMP<n>	A value that, when reached in the associated up-counter for this <i>Trigger State</i> , causes a <i>Trigger Counter Comparison</i> match to occur.

### 3.10.8 External Mask registers

The External Mask registers are used to mask out specific *External Trigger Input Signals* for *Trigger Signal* comparisons.

The EXTMASK<n> register characteristics are:

**Usage constraints** Writing when CTRL.RUN = 1 results in improper operation.

**Configurations** Available in all configurations. EXTMASK4 is only available when NUM\_TRIG\_STATES = 5.

**Attributes** See [3.9 Trigger State register summary on page 3-55](#).

The following table shows the bit assignments.

**Table 3-25 EXTMASK<n> register bit assignments**

Bits	Name	Function
[7:2]	EXTTRIG	Mask <b>EXTTRIG[5:0]</b> signals. Each signal is masked by clearing the appropriate bit.  0b0 <i>External Trigger Input Signal</i> is masked and is not used in comparisons. 0b1 <i>External Trigger Input Signal</i> is not masked.
[1:0]	CTTRIGIN	Mask <b>CTTRIGIN[1:0]</b> signals. Each signal is masked by clearing the appropriate bit.  0b0 <i>External Trigger Input Signal</i> is masked and is not used in comparisons. 0b1 <i>External Trigger Input Signal</i> is not masked.

#### Related concepts

[2.5 Triggering on page 2-33](#).

#### Related references

[3.10.10 Signal Mask registers on page 3-65](#).

[3.10.11 Signal Compare registers on page 3-66](#).

### 3.10.9 External Compare registers

The External Compare registers provide the data values with which the *External Trigger Input Signals* are compared.



The EXTCOMP<n> register characteristics are:

<b>Usage constraints</b>	Writing when CTRL.RUN = 1 results in improper operation.
<b>Configurations</b>	Available in all configurations. EXTCOMP4 is only available when NUM_TRIG_STATES = 5.
<b>Attributes</b>	See <a href="#">3.9 Trigger State register summary on page 3-55</a> .

The following table shows the bit assignments.

**Table 3-26 EXTCOMP<n> register bit assignments**

Bits	Name	Function
[7:2]	EXTTRIG	Compare value for <b>EXTTRIG[5:0]</b> signals.
[1:0]	CTTRIGIN	Compare value for <b>CTTRIGIN[1:0]</b> signals.

#### Related concepts

[2.5 Triggering on page 2-33](#).

#### Related references

[3.10.10 Signal Mask registers on page 3-65](#).

[3.10.11 Signal Compare registers on page 3-66](#).

### 3.10.10 Signal Mask registers

The Signal Mask registers are used to mask out specific *Signal Group* signals for *Trigger Signal* comparisons.

The SIGMASK<n> register characteristics are:

<b>Usage constraints</b>	Writing when CTRL.RUN = 1 results in improper operation.
<b>Configurations</b>	Available in all configurations. SIGMASK4 is only available when NUM_TRIG_STATES = 5.
<b>Attributes</b>	See <a href="#">3.9 Trigger State register summary on page 3-55</a> .

The following table shows the bit assignments.

#### Note

Each signal in the *Signal Group* is masked by clearing the appropriate bit.

**Table 3-27 SIGMASK<n> register bit assignments**

Bits	Name	Function
[31:0]	SIGMASK[31:0]	Mask bits from SIGCOMP[31:0].
[63:32]	SIGMASK[63:32]	Mask bits from SIGCOMP[63:32].
[95:64]	SIGMASK[95:64]	Mask bits from SIGCOMP[95:64]. These bits are only used if GRP_WIDTH = 128 or 256.
[127:96]	SIGMASK[127:96]	Mask bits from SIGCOMP[127:96]. These bits are only used if GRP_WIDTH = 128 or 256.

#### Related concepts

[2.5 Triggering on page 2-33](#).

#### Related references

[3.10.11 Signal Compare registers on page 3-66](#).

### 3.10.11 Signal Compare registers

The Signal Compare registers provide the data values with which the *Signal Group* signals are compared.

The SIGCOMP<n> register characteristics are:

**Usage constraints** Writing when CTRL.RUN = 1 results in improper operation.

**Configurations** Available in all configurations. SIGCOMP4 is only available when NUM\_TRIG\_STATES = 5.

**Attributes** See [3.9 Trigger State register summary on page 3-55](#).

The following table shows the bit assignments.

**Table 3-28 SIGCOMP<n> register bit assignments**

Bits	Name	Function
[31:0]	SIGCOMP[31:0]	Compare value for <i>Signal Group</i> signals[31:0].
[63:32]	SIGCOMP[63:32]	Compare value for <i>Signal Group</i> signals[63:32].
[95:64]	SIGCOMP[95:64]	Compare value for <i>Signal Group</i> signals[95:64]. These bits are only used if GRP_WIDTH = 128 or 256.
[127:96]	SIGCOMP[127:96]	Compare value for <i>Signal Group</i> signals[127:96]. These bits are only used if GRP_WIDTH = 128 or 256.

#### Related concepts

[2.5 Triggering on page 2-33](#).

#### Related references

[3.10.10 Signal Mask registers on page 3-65](#).

## 3.11 Integration Mode register summary

This section gives a summary of the ELA-500 Integration Mode registers.

The following table shows the integration mode registers in offset order from the base address of the ELA-500.

**Table 3-29 Integration mode registers summary**

Offset	Name	Type	Reset	Description
0xEE8	ITTRIGOUT	WO	0x00	<a href="#">3.12.1 Integration Mode Action Trigger Output register on page 3-68</a>
0xEF8	ITTRIGIN	RO	-	<a href="#">3.12.2 Integration Mode External Trigger Input register on page 3-68</a>
0xF00	ITCTRL	RW	0b0	<a href="#">3.12.3 Integration Mode Control register on page 3-69</a>

## 3.12 Integration Mode register descriptions

This section describes the ELA-500 Integration Mode registers.

[Table 3-29 Integration mode registers summary on page 3-67](#) provides cross-references to individual registers.

This section contains the following subsections:

- [3.12.1 Integration Mode Action Trigger Output register on page 3-68.](#)
- [3.12.2 Integration Mode External Trigger Input register on page 3-68.](#)
- [3.12.3 Integration Mode Control register on page 3-69.](#)

### 3.12.1 Integration Mode Action Trigger Output register

The Integration Mode Action Trigger Output register drives values on the *Output Actions*.

The ITTRIGOUT register characteristics are:

**Usage constraints** No access when CTRL.RUN = 1.

**Configurations** Available in all configurations.

**Attributes** See [3.11 Integration Mode register summary on page 3-67](#).

The following table shows the bit assignments.

**Table 3-30 ITTRIGOUT register bit assignments**

Bits	Name	Function
[7:4]	ELAOUTPUT	Value to drive on <b>ELAOUTPUT[3:0]</b> when ITCTRL.IME = 1.
[3]	Reserved	-
[2]	STOPCLOCK	Level to drive on <b>STOPCLOCK</b> when ITCTRL.IME = 1. 0b0 Drive 0 on <b>STOPCLOCK</b> . 0b1 Drive 1 on <b>STOPCLOCK</b> .
[1:0]	CTTRIGOUT	Value to drive on <b>CTTRIGOUT[1:0]</b> when ITCTRL.IME = 1.

### 3.12.2 Integration Mode External Trigger Input register

The Integration Mode External Trigger Input register captures the values on the eight trigger inputs.

The ITTRIGIN register characteristics are:

**Usage constraints** No access when CTRL.RUN = 1.

**Configurations** Available in all configurations.

**Attributes** See [3.11 Integration Mode register summary on page 3-67](#).

The following table shows the bit assignments.

**Table 3-31 ITTRIGIN register bit assignments**

Bits	Name	Function
[7:2]	EXTTRIG	Captures the value on <b>EXTTRIG[5:0]</b> when ITCTRL.IME = 1.
[1:0]	CTTRIGIN	Captures the value on <b>CTTRIGIN[1:0]</b> when ITCTRL.IME = 1.

**Note**

If the parameter `TRIGIN_EDGE = 0`, then **CTTRIGIN** and **EXTTRIG** must be held steady when reading **ITTRIGIN**. If `TRIGIN_EDGE = 1`, and if `ICTRL.IME = 1`, a rising-edge on **CTTRIGIN** or **EXTTRIGIN** is latched until `ICTLR.IME = 0`.

### 3.12.3 Integration Mode Control register

The Integration Mode control register enables testing of the eight external trigger inputs and the eight output actions.

The `ITCTRL` register characteristics are:

**Usage constraints** No access when `CTRL.RUN = 1`.

**Configurations** Available in all configurations.

**Attributes** See [3.11 Integration Mode register summary on page 3-67](#).

The following table shows the bit assignments.

**Table 3-32 ITCTRL register bit assignments**

Bits	Name	Function
[0]	IME	Integration Mode enable.
	0b0	Integration Mode disabled. The ELA-500 operates normally.
	0b1	Integration Mode enabled when <code>CTRL.RUN = 0</code> .

### 3.13 Software Lock register summary

This section gives a summary of the ELA-500 Software Lock registers.

The following table shows the software lock registers in offset order from the base address of the ELA-500.

**Table 3-33 Software lock registers summary**

Offset	Name	Type	Reset	Description
0xFB0	LAR	WO	-	<a href="#">3.14.1 Lock Access Register on page 3-71</a>
0xFB4	LSR	RO	0b000 or 0b011 <sup>c</sup>	<a href="#">3.14.2 Lock Status Register on page 3-71</a>

<sup>c</sup> The LSR reset value depends on whether the register is read by a processor or the debugger. If read by a processor, the reset value is 0b011. If read by a debugger it is 0b000.

## 3.14 Software Lock register descriptions

This section describes the ELA-500 Software Lock registers.

[Table 3-33 Software lock registers summary on page 3-70](#) provides cross-references to individual registers.

This section contains the following subsections:

- [3.14.1 Lock Access Register on page 3-71.](#)
- [3.14.2 Lock Status Register on page 3-71.](#)

### 3.14.1 Lock Access Register

The Lock Access Register controls write access for self-hosted, on-chip accesses to the ELA-500 registers.

The LAR characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.13 Software Lock register summary on page 3-70</a> .

The following table shows the bit assignments.

**Table 3-34 LAR bit assignments**

Bits	Name	Function
[31:0]	LAR	Permits writes to the other ELA-500 registers when the access code 0xC5ACCE55 is written. Writing any other value prevents access to the other ELA-500 registers.

### 3.14.2 Lock Status Register

The Lock Status Register returns the status of the lock access control.

The LSR characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.13 Software Lock register summary on page 3-70</a> .

The following table shows the bit assignments.

**Table 3-35 LSR bit assignments**

Bits	Name	Function
[2:0]	LSR	Returns the status of the lock access control.
		0b001 Write access permitted.
		0b011 Write access not permitted.

See the *ARM® CoreSight™ Architecture Specification* for more information.

## 3.15 Authentication register summary

This section gives a summary of the ELA-500 Authentication registers.

The following table shows the Authentication registers in offset order from the base address of the ELA-500.

**Table 3-36 Authentication registers summary**

Offset	Name	Type	Reset	Description
0xFB8	AUTHSTATUS	RO	-	<a href="#">3.16.1 Authentication Status register on page 3-73</a>



## 3.16 Authentication register descriptions

This section describes the ELA-500 Authentication registers.

[Table 3-36 Authentication registers summary on page 3-72](#) provides cross-references to individual registers.

### 3.16.1 Authentication Status register

The Authentication Status register returns the status of the authentication signals **DBGEN**, **NIDEN**, **SPIDEN**, and **SPNIDEN**.

The AUTHSTATUS register characteristics are:

**Usage constraints** No usage constraints.

**Configurations** Available in all configurations.

**Attributes** See [3.15 Authentication register summary on page 3-72](#).

The following table shows the bit assignments.

**Table 3-37 AUTHSTATUS bit assignments**

Bits	Name	Function
[7:6]	SNID	Secure, non-invasive debug.  0b10 Debug disabled. 0b11 Debug enabled.
[5:4]	SID	Secure, invasive debug.  0b10 Debug disabled. 0b11 Debug enabled.
[3:2]	NSNID	Non-secure, non-invasive debug.  0b10 Debug disabled. 0b11 Debug enabled.
[1:0]	NSID	Non-secure, invasive debug.  0b10 Debug disabled. 0b11 Debug enabled.

See the *ARM® CoreSight™ Architecture Specification* for more information.

#### Related concepts

[2.6 Authentication interface on page 2-38](#).

## 3.17 Device register summary

This section gives a summary of the ELA-500 Device registers.

The following table shows the device registers in offset order from the base address of the ELA-500.

<sup>d</sup>

**Table 3-38 Device registers summary**

Offset	Name	Type	Reset	Description
0xFBC	DEVARCH	RO	0x47700A75	<a href="#">3.18.1 Device Architecture register on page 3-75</a>
0xFC0	DEVID2	RO	Configuration-dependent	<a href="#">3.18.2 Device Configuration register 2 on page 3-75</a>
0xFC4	DEVID1	RO	Configuration-dependent	<a href="#">3.18.3 Device Configuration register 1 on page 3-76</a>
0xFC8	DEVID	RO	Configuration-dependent	<a href="#">3.18.4 Device Configuration register on page 3-76</a>
0xFCC	DEVTYPE	RO	0x75	<a href="#">3.18.5 Device Type Identifier register on page 3-77</a>

<sup>d</sup> Configuration-dependent.

## 3.18 Device register descriptions

This section describes the ELA-500 Device registers.

[Table 3-38 Device registers summary on page 3-74](#) provides cross-references to individual registers.

This section contains the following subsections:

- [3.18.1 Device Architecture register on page 3-75.](#)
- [3.18.2 Device Configuration register 2 on page 3-75.](#)
- [3.18.3 Device Configuration register 1 on page 3-76.](#)
- [3.18.4 Device Configuration register on page 3-76.](#)
- [3.18.5 Device Type Identifier register on page 3-77.](#)

### 3.18.1 Device Architecture register

The Device Architecture register returns the architect and architecture of the ELA-500.

The DEVARCH register characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.17 Device register summary on page 3-74</a> .

The following table shows the bit assignments.

**Table 3-39 DEVARCH bit assignments**

Bits	Name	Function
[31:21]	ARCHITECT	The architect of the device. 0x23B ARM.
[20]	PRESENT	Indicates that the register is present. 1 Register present.
[19:16]	REVISION	Architecture revision. 0 First revision.
[15:0]	ARCHID	The architecture of the device. 0x0A75 CoreSight ELA.

See the *ARM® CoreSight™ Architecture Specification* for more information.

### 3.18.2 Device Configuration register 2

The Device Configuration register 2 provides configuration information about the ELA-500.

The DEVID2 register characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.17 Device register summary on page 3-74</a> .

The following table shows the bit assignments.

**Table 3-40 DEVID2 bit assignments**

Bits	Name	Function
[31:20]	-	Reserved
[19:16]	TRIGIN_EDGE	<p>0 Level detect of <b>CTTRIGIN</b> and <b>EXTTRIG</b>.</p> <p>1 Single edge detect of <b>CTTRIGIN</b> and <b>EXTTRIG</b>.</p>
[15:8]	COMP_WIDTH	<p>Indicates the comparator width.</p> <p>0 Comparator width = GRP_WIDTH.</p> <p>&gt;0 Comparator width = (COMP_WIDTH + 1) x 8.</p> <p>For example, if COMP_WIDTH = 15, then comparator width = 128.</p>
[7:0]	ALTTS	<p>Indicates whether <i>Alternate Trace Select</i> (ALTTS) is implemented or not.</p> <p>0x00 ALTTS is not implemented.</p> <p>0x10 ALTTS is implemented and can be used for independent trace of <i>Trigger State 4</i>.</p> <p>All other encodings are reserved and read as 0x00.</p> <p>See <a href="#">2.4.3 Second trace comparator on Trigger State 4 on page 2-29</a> for more information.</p>

### 3.18.3 Device Configuration register 1

The Device Configuration register 1 provides configuration information about the ELA-500.

The DEVID1 register characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.17 Device register summary on page 3-74</a> .

The following table shows the bit assignments.

**Table 3-41 DEVID1 bit assignments**

Bits	Name	Function
[31:24]	COUNTWIDTH	Counter width in bits. Fixed at 32.
[23:16]	NUMTRIGSTATES	Number of <i>Trigger States</i> . Four or five.
[15:8]	SIGGRPWIDTH	<p><i>Signal Group</i> width. The field value is (<i>Signal Group</i> width/8) - 1.</p> <p>For example, 7 if GRP_WIDTH = 64, 15 if GRP_WIDTH = 128, and 31 if GRP_WIDTH = 256.</p>
[7:0]	NUMSIGGRPS	Number of <i>Signal Groups</i> . Fixed at 12.

### 3.18.4 Device Configuration register

The Device Configuration register provides configuration information about the ELA-500.

The DEVID register characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.17 Device register summary on page 3-74</a> .

The following table shows the bit assignments.

**Table 3-42 DEVID bit assignments**

Bits	Name	Function
[31:29]	-	Reserved
[28:25]	SCRAMBLER	<p>0 Trace read data scrambler not present.</p> <p>1 Trace read data scrambler present.</p>
[24:20]	ID_CAPTURE_SIZE	2-30 bits when COND_TRIG = 1, or 0 otherwise.
[19:16]	COND_TRIG	<p>Shows the value of the COND_TRIG parameter.</p> <p>1, when COND_TRIG = 1, or 0 otherwise.</p>
[15:8]	SRAM_ADDR_SIZE	SRAM address width in bits.
[7:4]	TRACEFORMAT	<p>Trace implementation:</p> <p>1 Fixed at 1. Indicates Trace header format revision 1.</p>
[3:0]	TRACETYPE	<p>ATB trace:</p> <p>0 ATB trace not implemented.</p> <p>1 ATB trace is implemented.</p>

### 3.18.5 Device Type Identifier register

The Device Type Identifier register returns the device type.

The DEVTYPE register characteristics are:

**Usage constraints** No usage constraints.

**Configurations** Available in all configurations.

**Attributes** See [3.17 Device register summary on page 3-74](#).

The following table shows the bit assignments.

**Table 3-43 DEVTYPE bit assignments**

Bits	Name	Function
[7:0]	DEVTYPE	<p>0x75.</p> <p>SUB type = 0x7.</p> <p>MAJOR type = 0x5.</p>

## 3.19 ID register summary

This section gives a summary of the ELA-500 ID registers.

The following table shows the device registers in offset order from the base address of the ELA-500.

**Table 3-44 ID registers summary**

Offset	Name	Type	Reset	Description
0xFD0	PIDR4	RO	0x04	<a href="#">3.20.1 Peripheral ID4 Register on page 3-79</a>
0xFD4	PIDR5	RO	0x00	<a href="#">3.20.2 Peripheral ID5 Register on page 3-79</a>
0xFD8	PIDR6	RO	0x00	<a href="#">3.20.3 Peripheral ID6 Register on page 3-79</a>
0xFDC	PIDR7	RO	0x00	<a href="#">3.20.4 Peripheral ID7 Register on page 3-80</a>
0xFE0	PIDR0	RO	0xB8	<a href="#">3.20.5 Peripheral ID0 Register on page 3-80</a>
0xFE4	PIDR1	RO	0xB9	<a href="#">3.20.6 Peripheral ID1 Register on page 3-80</a>
0xFE8	PIDR2	RO	0x3B	<a href="#">3.20.7 Peripheral ID2 Register on page 3-81</a>
0xFEC	PIDR3	RO	0x00	<a href="#">3.20.8 Peripheral ID3 Register on page 3-81</a>
0xFF0	CIDR0	RO	0x0D	<a href="#">3.20.9 Component ID0 Register on page 3-81</a>
0xFF4	CIDR1	RO	0x90	<a href="#">3.20.10 Component ID1 Register on page 3-82</a>
0xFF8	CIDR2	RO	0x05	<a href="#">3.20.11 Component ID2 Register on page 3-82</a>
0xFFC	CIDR3	RO	0xB1	<a href="#">3.20.12 Component ID3 Register on page 3-82</a>

## 3.20 ID register descriptions

This section describes the ELA-500 ID registers.

[Table 3-44 ID registers summary on page 3-78](#) provides cross-references to individual registers.

This section contains the following subsections:

- [3.20.1 Peripheral ID4 Register on page 3-79.](#)
- [3.20.2 Peripheral ID5 Register on page 3-79.](#)
- [3.20.3 Peripheral ID6 Register on page 3-79.](#)
- [3.20.4 Peripheral ID7 Register on page 3-80.](#)
- [3.20.5 Peripheral ID0 Register on page 3-80.](#)
- [3.20.6 Peripheral ID1 Register on page 3-80.](#)
- [3.20.7 Peripheral ID2 Register on page 3-81.](#)
- [3.20.8 Peripheral ID3 Register on page 3-81.](#)
- [3.20.9 Component ID0 Register on page 3-81.](#)
- [3.20.10 Component ID1 Register on page 3-82.](#)
- [3.20.11 Component ID2 Register on page 3-82.](#)
- [3.20.12 Component ID3 Register on page 3-82.](#)

### 3.20.1 Peripheral ID4 Register

The Peripheral ID4 Register returns byte[4] of the Peripheral ID.

The PIDR4 characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.19 ID register summary on page 3-78.</a>

The following table shows the bit assignments.

**Table 3-45 PIDR4 bit assignments**

Bits	Name	Function
[7:4]	SIZE	0x0. One 4KB count.
[3:0]	DES_2	0x4. JEP continuation code for ARM.

### 3.20.2 Peripheral ID5 Register

The Peripheral ID5 Register returns byte[5] of the Peripheral ID.

The PIDR5 characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.19 ID register summary on page 3-78.</a>

The following table shows the bit assignments.

**Table 3-46 PIDR5 bit assignments**

Bits	Name	Function
[7:0]	PIDR5	0x00. Reserved.

### 3.20.3 Peripheral ID6 Register

The Peripheral ID6 Register returns byte[6] of the Peripheral ID.

The PIDR6 characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.19 ID register summary on page 3-78</a> .

The following table shows the bit assignments.

**Table 3-47 PIDR6 bit assignments**

Bits	Name	Function
[7:0]	PIDR6	0x00. Reserved.

### 3.20.4 Peripheral ID7 Register

The Peripheral ID7 Register returns byte[7] of the Peripheral ID.

The PIDR7 characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.19 ID register summary on page 3-78</a> .

The following table shows the bit assignments.

**Table 3-48 PIDR7 bit assignments**

Bits	Name	Function
[7:0]	PIDR7	0x00. Reserved.

### 3.20.5 Peripheral ID0 Register

The Peripheral ID0 Register returns byte[0] of the Peripheral ID.

The PIDR0 characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.19 ID register summary on page 3-78</a> .

The following table shows the bit assignments.

**Table 3-49 PIDR0 bit assignments**

Bits	Name	Function
[7:0]	PART_0	0xB8. Bits[7:0] of part number 0x9B8.

### 3.20.6 Peripheral ID1 Register

The Peripheral ID1 Register returns byte[1] of the Peripheral ID.

The PIDR1 characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.19 ID register summary on page 3-78</a> .

The following table shows the bit assignments.



**Table 3-50 PIDR1 bit assignments**

Bits	Name	Function
[7:4]	DES_0	0xB. Bits[3:0] of JEP106 identification code for ARM 0x3B.
[3:0]	PART_1	0x9. Bits[11:8] of part number 0x9B8.

### 3.20.7 Peripheral ID2 Register

The Peripheral ID2 Register returns byte[2] of the Peripheral ID.

The PIDR2 characteristics are:

**Usage constraints** No usage constraints.  
**Configurations** Available in all configurations.  
**Attributes** See [3.19 ID register summary on page 3-78](#).

The following table shows the bit assignments.

**Table 3-51 PIDR2 bit assignments**

Bits	Name	Function
[7:4]	REVISION	0x3. Revision number. Indicates revision r2p1.
[3]	JEDEC	0b1. Fixed at 0b1.
[2:0]	DES_1	0b011. Bits[6:4] of JEP106 identification code for ARM 0x3B.

### 3.20.8 Peripheral ID3 Register

The Peripheral ID3 Register returns byte[3] of the Peripheral ID.

The PIDR3 characteristics are:

**Usage constraints** No usage constraints.  
**Configurations** Available in all configurations.  
**Attributes** See [3.19 ID register summary on page 3-78](#).

The following table shows the bit assignments.

**Table 3-52 PIDR3 bit assignments**

Bits	Name	Function
[7:4]	REVAND	0x0. RevAnd.
[3:0]	CMOD	0x0. Indicates whether the customer has modified the behavior of the component. In most cases, this field is 0000. You can change this value when you make authorized modifications to this component.

### 3.20.9 Component ID0 Register

The Component ID0 Register returns byte[0] of the Component ID.

The CIDR0 characteristics are:

**Usage constraints** No usage constraints.  
**Configurations** Available in all configurations.  
**Attributes** See [3.19 ID register summary on page 3-78](#).

The following table shows the bit assignments.

**Table 3-53 CIDR0 bit assignments**

Bits	Name	Function
[7:0]	PRMBL_0	0x0D. Preamble.

### 3.20.10 Component ID1 Register

The Component ID1 Register returns byte[1] of the Component ID.

The CIDR1 characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.19 ID register summary on page 3-78</a> .

The following table shows the bit assignments.

**Table 3-54 CIDR1 bit assignments**

Bits	Name	Function
[7:4]	CLASS	0x9. Indicates a CoreSight component.
[3:0]	PRMBL_1	0x0. Preamble.

### 3.20.11 Component ID2 Register

The Component ID2 Register returns byte[2] of the Component ID.

The CIDR2 characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.19 ID register summary on page 3-78</a> .

The following table shows the bit assignments.

**Table 3-55 CIDR2 bit assignments**

Bits	Name	Function
[7:0]	PRMBL_2	0x05. Preamble.

### 3.20.12 Component ID3 Register

The Component ID3 Register returns byte[3] of the Component ID.

The CIDR3 characteristics are:

<b>Usage constraints</b>	No usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See <a href="#">3.19 ID register summary on page 3-78</a> .

The following table shows the bit assignments.

**Table 3-56 CIDR3 bit assignments**

Bits	Name	Function
[7:0]	PRMBL_3	0xB1. Preamble.

# Appendix A

## Signal descriptions

This appendix describes the external signals of the ELA-500.

It contains the following sections:

- *A.1 Clocks and reset* on page Appx-A-84.
- *A.2 Debug APB signals* on page Appx-A-85.
- *A.3 Observation interface signals* on page Appx-A-86.
- *A.4 Timestamp interface signals* on page Appx-A-87.
- *A.5 Authentication interface signals* on page Appx-A-88.
- *A.6 DFT and MBIST interface signals* on page Appx-A-89.
- *A.7 Q-Channel Low-Power interface signals* on page Appx-A-90.
- *A.8 Output Action signals* on page Appx-A-91.
- *A.9 External Trigger Input signals* on page Appx-A-92.

## A.1 Clocks and reset

The ELA-500 has several clock and reset signals.

The following table shows the ELA-500 clock and reset signals.

**Table A-1 ELA-500 clock and reset signals**

Signal name	Type	Description
<b>ELACK</b>	Input	Logic analyzer clock for triggering and trace.
<b>RESETn</b>		Reset for <b>ELACK</b> domain including <i>Output Actions</i> .
<b>PCLKDBG</b>		Clock for debug APB interface.
<b>PRESETDBGn</b>		Reset for debug APB domain and operation.

## A.2 Debug APB signals

The following table shows the ELA-500 debug APB signals. All the signals are in the **PCLKDBG** clock domain.

**Table A-2 ELA-500 debug APB signals**

Signal name	Type	Description	Connection information
<b>PSELDBG</b>	Input	Select.	Connect to the CoreSight debug subsystem.
<b>PENABLEDBG</b>		Enable.	
<b>PWRITEDBG</b>		Peripheral write.	
<b>PADDRDBG[11:2]</b>		Address. The ELA-500 only supports word-aligned addresses.	
<b>PADDRDBG31</b>		Indicates the source of a debug APB access. LOW when accesses originate from software running on a processor within the system, for example self-hosted debug software, and HIGH when accesses originate from an external debugger.	
<b>PWDATADBG[31:0]</b>	Output	Write data.	
<b>PREADYDBG</b>		Peripheral ready.	
<b>PSLVERRDBG</b>		Slave error.	
<b>PRDATADBG[31:0]</b>		Read data.	

### A.3 Observation interface signals

The following table shows the ELA-500 Observation interface signals. All the signals are in the **ELACLK** clock domain.

**Note**

The number of signals in each SIGNALGRP can be 64, 128, or 256, depending on GRP\_WIDTH.

**Table A-3 ELA-500 Observation interface signals**

Signal name	Type	Description	Connection information
<b>SIGNALGRP0</b> [GRP_WIDTH-1:0]	Input	Signal Group 0 debug signals.	Connect to IP debug signals.
<b>SIGCLKEN0</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP0</b> .	
<b>SIGNALGRP1</b> [GRP_WIDTH-1:0]		Signal Group 1 debug signals.	
<b>SIGCLKEN1</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP1</b> .	
<b>SIGNALGRP2</b> [GRP_WIDTH-1:0]		Signal Group 2 debug signals.	
<b>SIGCLKEN2</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP2</b> .	
<b>SIGNALGRP3</b> [GRP_WIDTH-1:0]		Signal Group 3 debug signals.	
<b>SIGCLKEN3</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP3</b> .	
<b>SIGNALGRP4</b> [GRP_WIDTH-1:0]		Signal Group 4 debug signals.	
<b>SIGCLKEN4</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP4</b> .	
<b>SIGNALGRP5</b> [GRP_WIDTH-1:0]		Signal Group 5 debug signals.	
<b>SIGCLKEN5</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP5</b> .	
<b>SIGNALGRP6</b> [GRP_WIDTH-1:0]		Signal Group 6 debug signals.	
<b>SIGCLKEN6</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP6</b> .	
<b>SIGNALGRP7</b> [GRP_WIDTH-1:0]		Signal Group 7 debug signals.	
<b>SIGCLKEN7</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP7</b> .	
<b>SIGNALGRP8</b> [GRP_WIDTH-1:0]		Signal Group 8 debug signals.	
<b>SIGCLKEN8</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP8</b> .	
<b>SIGNALGRP9</b> [GRP_WIDTH-1:0]		Signal Group 9 debug signals.	
<b>SIGCLKEN9</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP9</b> .	
<b>SIGNALGRP10</b> [GRP_WIDTH-1:0]		Signal Group 10 debug signals.	
<b>SIGCLKEN10</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP10</b> .	
<b>SIGNALGRP11</b> [GRP_WIDTH-1:0]		Signal Group 11 debug signals.	
<b>SIGCLKEN11</b>		<b>ELACLK</b> clock enable for <b>SIGNALGRP11</b> .	

## A.4 Timestamp interface signals

The following table shows the ELA-500 timestamp interface signals. The input value must be synchronized into the **ELACLK** clock domain.

**Table A-4 ELA-500 timestamp interface signals**

Signal name	Type	Description	Connection information
TSVALUE[63:0]	Input	Timestamp value, encoded as a natural binary number.	From Timestamp Generator or decoder.

## A.5 Authentication interface signals

The following table shows the ELA-500 Authentication interface signals. All the signals must be synchronized into the **PCLKDBG** clock domain.

**Table A-5 ELA-500 Authentication interface signals**

Signal name	Type	Description	Connection information
<b>DBGEN</b>	Input	Invasive debug enable.	From CoreSight debug subsystem authentication control signals.
<b>NIDEN</b>		Non-invasive debug enable.	
<b>SPIDEN</b>		Secure invasive debug enable.	
<b>SPNIDEN</b>		Secure non-invasive debug enable.	



## A.6 DFT and MBIST interface signals

The following table shows the ELA-500 SRAM BIST interface signals. All the signals are in the **ELACLK** clock domain.

**Note**

All outputs are tied to zero when the configuration parameter **TRACE\_GEN** == 0.

**Table A-6 ELA-500 DFT and MBIST interface signals**

Signal name	Type	Description	Connection information
<b>DFTRAMHOLD</b>	Input	Disables the RAM chip select during scan shift.	Connect to scan DFT logic.
<b>MBISTREQ</b>		MBIST Mode Request. Enables MBIST testing.	Connect to MBIST controller.
<b>nMBISTRESET</b>		Resets functional logic to enable MBIST operations.	
<b>MBISTADDR[RAM_ADDR_SIZE-1:0]</b>		Logical RAM address.	
<b>MBISTINDATA[(GRP_WIDTH+8)-1:0]</b>		RAM Write data.	
<b>MBISTWRITEEN</b>		Write enable control. A no-op occurs if write and read enables are both zero.	
<b>MBISTREADEN</b>		Read enable control. A no-op occurs if write and read enables are both zero.	
<b>MBISTACK</b>	Output	MBIST Mode Ready. The ELA-500 acknowledges that it is MBIST-ready.	
<b>MBISTOUTDATA[(GRP_WIDTH+8)-1:0]</b>		RAM Read data.	

## A.7 Q-Channel Low-Power interface signals

The following table shows the ELA-500 Q-Channel Low-Power interface signals.

All the signals are in the **ELACLK** clock domain, except for **ELAQACTIVE**, which is generated by both **PCLKDBG** and **ELACLK**.

**Table A-7 ELA-500 Q-Channel Low-Power interface signals**

Signal name	Type	Description	Connection information
<b>ELAQREQn</b>	Input	Quiescence request from the clock controller to the ELA-500.	Connect to clock controller or power controller.
<b>ELAQACCEPTn</b>	Output	Quiescence request accept from the ELA-500.	
<b>ELAQDENY</b>		Quiescence request deny from the ELA-500.	
<b>ELAQACTIVE</b>		Indicates that the ELA-500 is active.	

## A.8 Output Action signals

The following table shows the ELA-500 *Output Action* signals. All the signals are in the **ELACLK** clock domain.

**Table A-8 ELA-500 *Output Action* signals**

Signal name	Type	Description	Connection information
CTTRIGOUT[1:0]	Output	Trigger outputs.	Connect to external CTI inputs.
STOPCLOCK		Used to stop SoC clocks.	Connect to SoC clock control.
ELAOUTPUT[3:0]		General-purpose outputs.	Connect to GIC, external I/O, or register.

## A.9 External Trigger Input signals

The following table shows the ELA-500 *External Trigger Input* Signals. All the signals must be synchronized into the **ELACLK** clock domain.

**Table A-9 ELA-500 External Trigger Input signals**

Signal name	Type	Description	Connection information
<b>CTTRIGIN</b> [1:0]	Input	Trigger inputs.	From external CTI or CoreSight EVENT interface.
<b>EXTTRIG</b> [5:0]		External inputs for non-debug signal <i>Trigger Condition</i> .	From other ELA-500 <b>ELAOUTPUT</b> signals or user-defined synchronized signals.

### Note

**CTTRIGIN** and **EXTTRIGIN** are compared with *Trigger Signal Comparisons*. It is recommended that a level is driven from the source connections to support the timing of the *Trigger Signal Comparison*. However, some cross-triggering sources might require detection of a one-time pulse.

Setting the parameter **TRIGIN\_EDGE** = 1 configures **CTTRIGIN** and **EXTTRIGIN** for detection of a single rising edge on each input when **CTRL.RUN** = 1 or **ICTRL.IME** = 1. When a rising-edge is detected, **CTTRIGIN** and **EXTTRIGIN** can only be cleared when **CTRL.RUN** = 0 or **ICTLR.IME** = 0 is written.

# Appendix B

## Revisions

This appendix describes the technical changes between released issues of this book.

It contains the following sections:

- [B.1 Revisions on page Appx-B-94.](#)

## B.1 Revisions

Each table lists the technical differences between successive issues of the document.

**Table B-1 Issue 0000-01**

Change	Location	Affects
First release	-	-

**Table B-2 Differences between issue 0000-01 and issue 0000-02**

Change	Location	Affects
Reset value for TIMECTRL register updated.	<a href="#">3.3 Control register summary on page 3-44</a>	r0p0
Usage constraints updated.	<a href="#">3.4.2 Timestamp Control register on page 3-45</a>	r0p0
	<a href="#">3.4.4 Pre-trigger Action register on page 3-46</a>	
	<a href="#">3.10.1 Signal Select registers on page 3-58</a>	
	<a href="#">3.10.2 Trigger Control registers on page 3-59</a>	
	<a href="#">3.10.3 Next State registers on page 3-61</a>	
	<a href="#">3.10.4 Action registers on page 3-61</a>	
	<a href="#">3.10.7 Counter Compare registers on page 3-64</a>	
	<a href="#">3.10.8 External Mask registers on page 3-64</a>	
	<a href="#">3.10.9 External Compare registers on page 3-64</a>	
	<a href="#">3.10.10 Signal Mask registers on page 3-65</a>	
	<a href="#">3.10.11 Signal Compare registers on page 3-66</a>	
Reset value for ACTION0, ACTION1, ACTION2, and ACTION3 registers updated.	<a href="#">3.9 Trigger State register summary on page 3-55</a>	r0p0

**Table B-3 Differences between issue 0000-02 and issue 0100-00**

Change	Location	Affects
Addition of r1p0 parameters table.	<a href="#">2.7 Parameter summary on page 2-39</a>	r1p0
Addition of DEVID2 register.	<a href="#">3.18.2 Device Configuration register 2 on page 3-75</a>	r1p0
Addition of FINALSTATE bit in CTSR.	<a href="#">3.6.1 Current Trigger State Register on page 3-49</a>	r1p0
DEVID.TRACEFORMAT fixed at 1.	<a href="#">3.18.4 Device Configuration register on page 3-76</a>	r1p0
PIDR2 revision ID changed to 1.	<a href="#">3.20.7 Peripheral ID2 Register on page 3-81</a>	r1p0
Addition of <i>Trigger Signal Alternative Comparison</i> state.	<a href="#">1.2 Definitions of terms used in this book on page 1-13</a> and throughout the document	r1p0
Addition of <i>Trigger State 4</i> .	Throughout the document	r1p0
Addition of Transaction ID capture.	Throughout the document	r1p0

**Table B-4 Differences between issue 0100-00 and issue 0200-00**

Change	Location	Affects
Addition of trace read data scrambler.	<a href="#">SRAM reads on page 2-32</a> , and throughout the document	r2p0
Addition of single rising-edge trigger configuration.	<a href="#">3.12.2 Integration Mode External Trigger Input register on page 3-68</a> , and throughout the document	r2p0
<b>MBISTREQ</b> no longer recognized on Low-power interface Q-Channel.	<a href="#">2.2 Interfaces on page 2-25</a>	r2p0
Addition of <b>TRIGIN_EDGE</b> parameter.	<a href="#">2.7 Parameter summary on page 2-39</a>	r2p0
Revision number increased to 2.	<a href="#">3.20.7 Peripheral ID2 Register on page 3-81</a>	r2p0
Width of Signal Groups increased to 256 bits. <b>GRP_WIDTH</b> increased accordingly.	Throughout the document	r2p0

**Table B-5 Differences between issue 0200-00 and issue 0201-00**

Change	Location	Affects
Revision number increased to 3.	<a href="#">3.20.7 Peripheral ID2 Register on page 3-81</a>	r2p1