ARM® CoreSight™ ELA-500 Embedded Logic Analyzer

Revision: r1p0

Technical Reference Manual

ARM

ARM 100127_0100_00_en

ARM® CoreSight™ ELA-500 Embedded Logic Analyzer

Technical Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-00-01	30 September 2014	Confidential	First draft for r0p0
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

Non-Confidential Proprietary Notice

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

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

It contains the following:

About this book

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

Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

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

p*n* 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 Embedded Logic Analyzer.

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 in its full configuration.

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.

`mono`space

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.

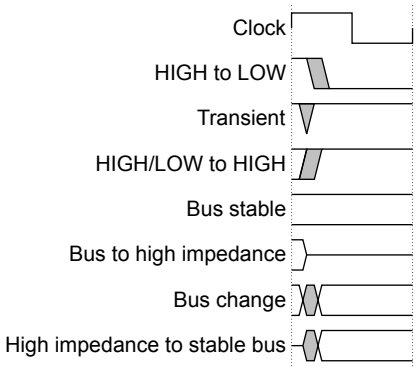

Figure 1 Key to timing diagram conventions

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.

Lower-case 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*. Give:

- The title *ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Technical Reference Manual*.
- The number ARM 100127_0100_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

The ELA-500 Embedded Logic Analyzer is a Design for Debug 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

The ELA-500 Embedded Logic Analyzer Technical Reference Manual uses specific terms. See this topic for a list of terms and their meanings.

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 either 64 bits or 128 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 Embedded Logic Analyzer 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 Embedded Logic Analyzer 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 Embedded Logic Analyzer 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 or 128 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 also 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.

This section contains the following subsections:

1.6.1 Configurable parameters

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

Related concepts

2.7 Parameter summary on page 2-38.

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:
  - The build configuration of the implementation.

    Note

    Build configuration information is also readable from the DEVID registers.

  - What integration, if any, was performed before implementing the ELA-500.
- 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 Embedded Logic Analyzer.

r0p0 First release.

r1p0 Adds new features to support the interconnect. These features are disabled by default with parameters. The ELA-500 Embedded Logic Analyzer is backward compatible with version 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.

Chapter 2
Functional description

This chapter describes the functionality of the ELA-500.

It contains the following sections:

2.1 About the functions

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

The ELA-500 can be configured to have either 64 or 128 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 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.

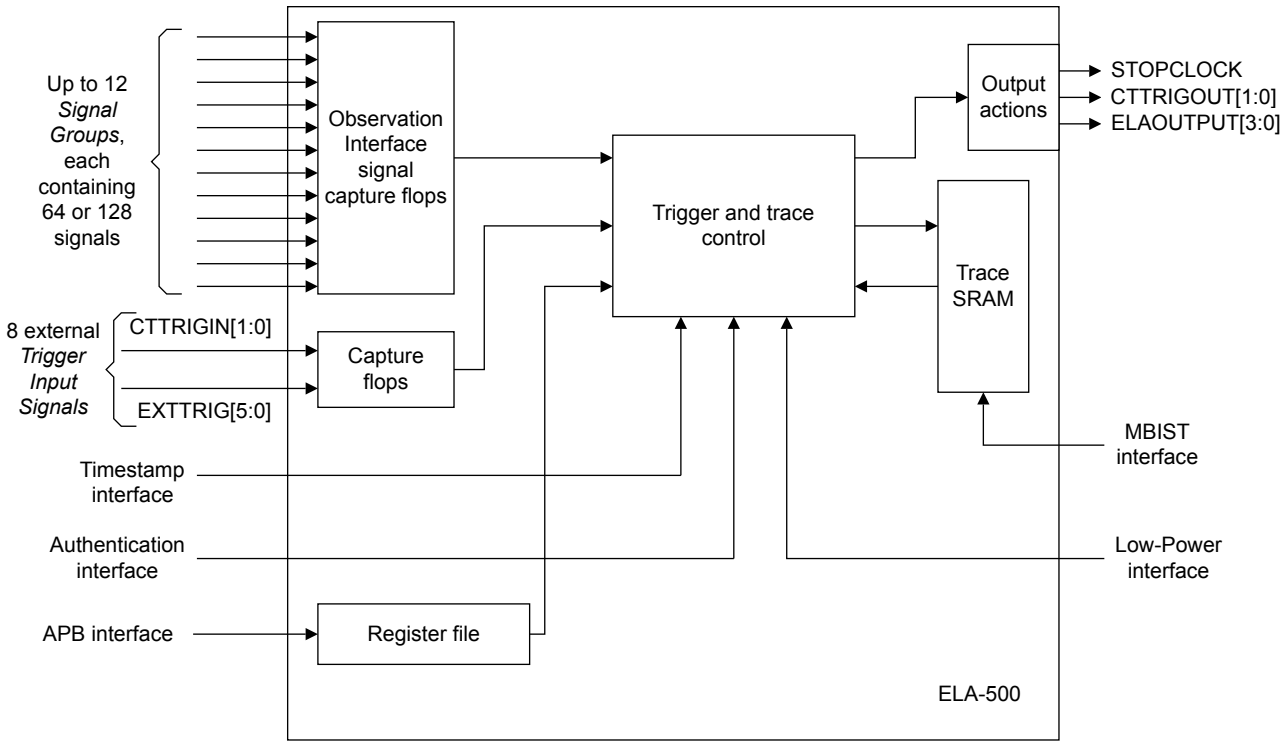

Figure 2-1 ELA-500 block diagram

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

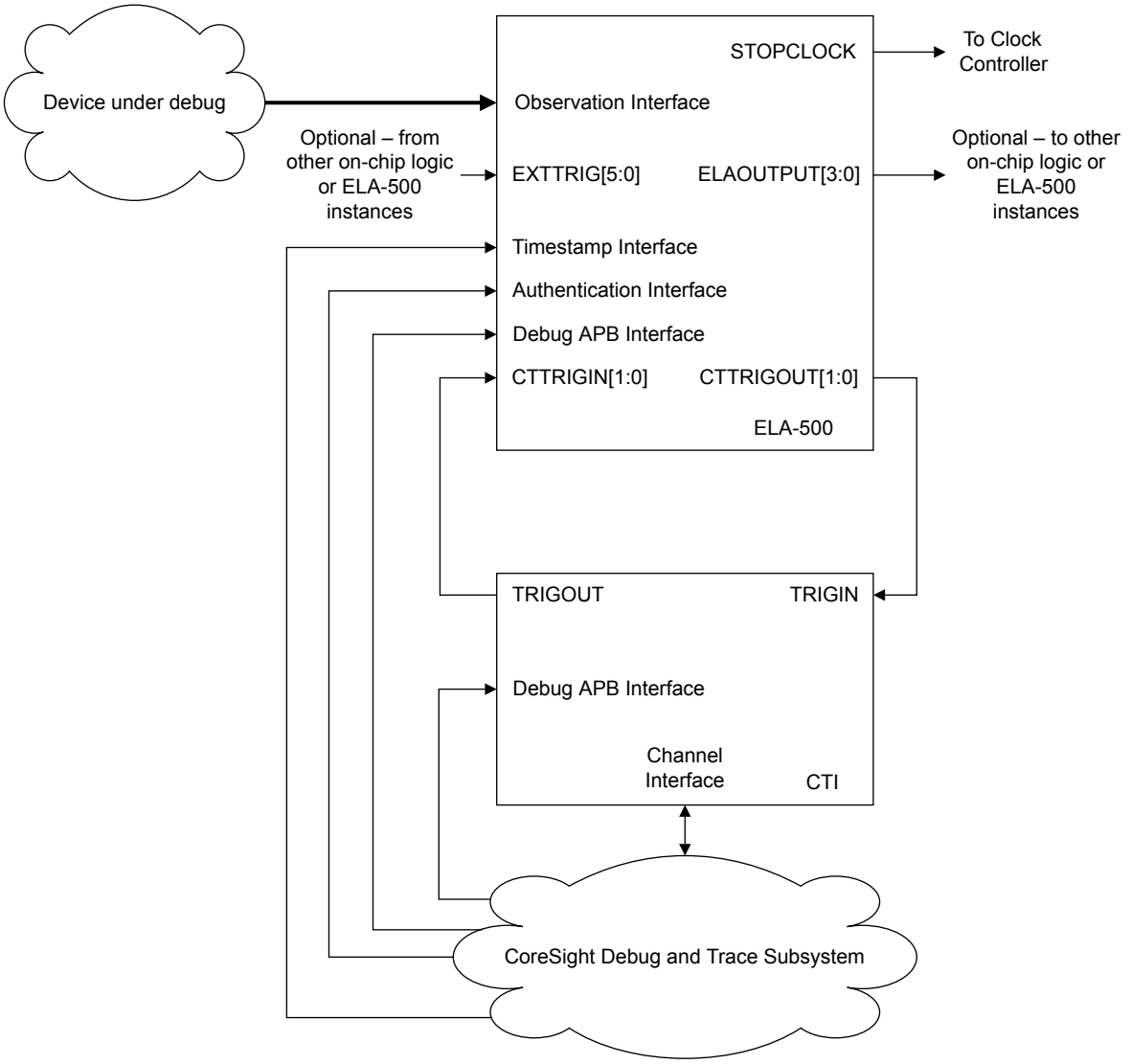

Figure 2-2 How to use the ELA-500 in a system

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 either 64 or 128 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:

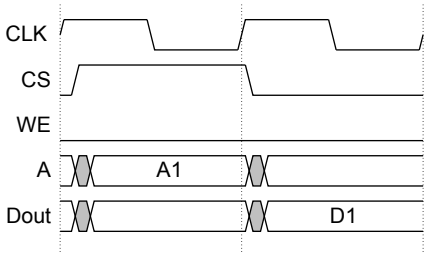

Figure 2-3 SRAM read access timing

The following figure shows the SRAM write access timing:

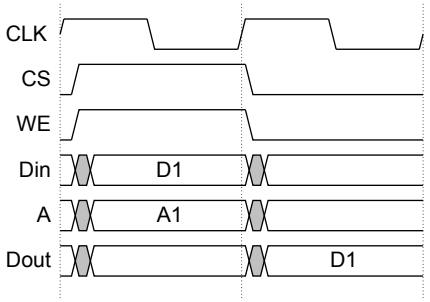

Figure 2-4 SRAM write access timing

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** = 0.
- There is no SRAM access in progress.
- **MBISTREQ** is not asserted by the MBIST controller.

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

- Unsynchronized **PSELDBG**.
- Unsynchronized **MBISTREQ**.
- SRAM busy, that indicates a one cycle read or write is in progress.
- CTRL.RUN = 1.

An internal active signal is generated with synchronized **MBISTREQ** and **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-88.
A.7 Q-Channel Low-Power interface signals on page Appx-A-89.

2.3 Clocking and reset

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

This section contains the following subsections:

2.3.1 Clocking

The ELA-500 logic analyzer 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:

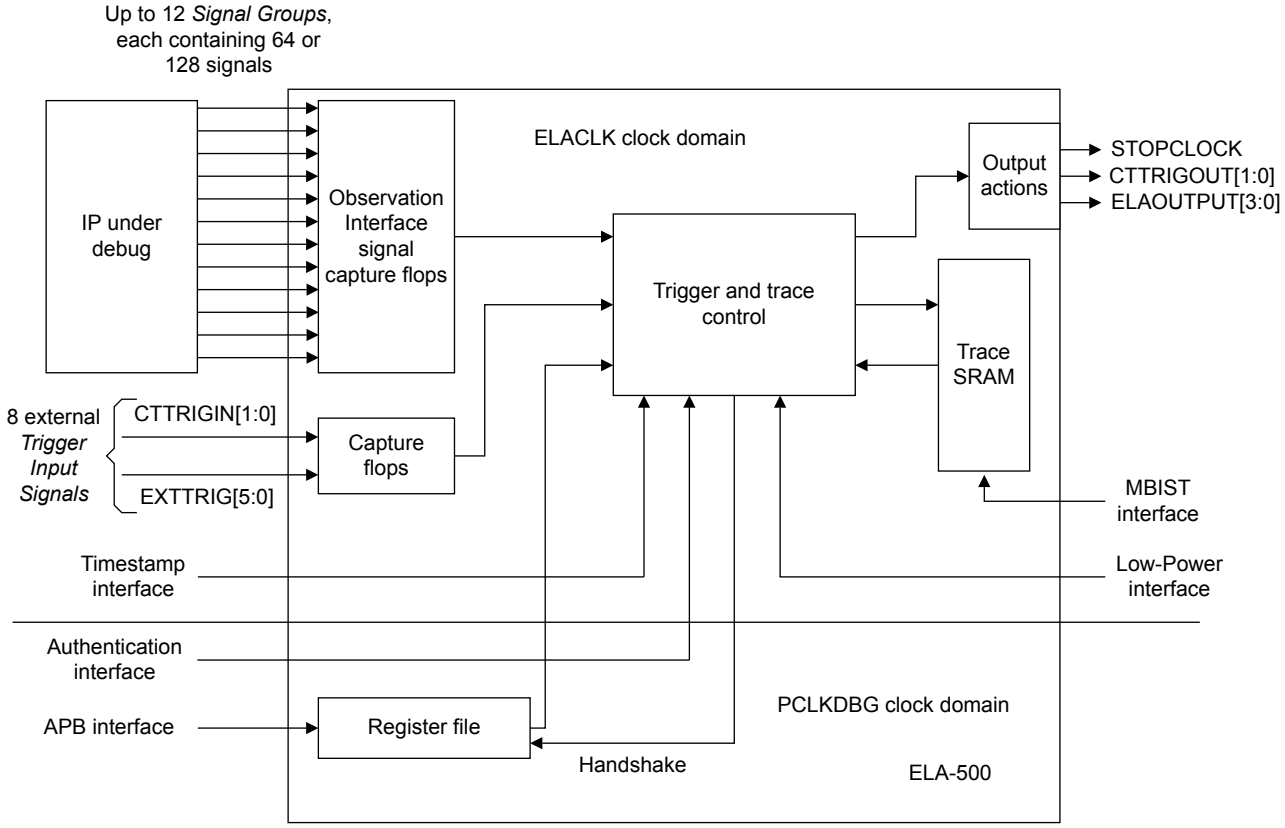

Figure 2-5 ELA-500 clock domains

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 be checked thoroughly in the SoC design to make sure that it does not affect the initialization of the SoC following a power-up reset.
- The **TSVALUE** signals in the Timestamp interface must be synchronized to **ELACLK**.

Related references

3.6 Current State register descriptions on page 3-48.

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

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 3-58.

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.

There is a new register, the *Trigger State Select Register* (TSSR), that is used to control 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 zero.
- 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 **ELACLK** cycle later, or if CTRL.RUN is cleared. If TRIGCTRL4.TRACESRC = 0, a trace is captured every six ELACLK cycles when COUNTCOMP4 = 5.

2.4.4 Trace SRAM format

The trace SRAM is used to capture a full *Signal Group* on every **ELACLK** 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]}
```

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 *Trigger State* 4 has overwritten data that was being written at the same time from a different *Trigger State*, that is *Trigger States* 0-3. Requires that TSSR.ALTTS4 = 1.

Table 2-1 Header byte format (continued)

Bits	Name	Function	
[4:2]	Trigger state	Current *Trigger State*. Software can use the *Trigger State* to determine which **SIGNALGRP<n>** was traced, by reading SIGSEL<n>.	
[1:0]	Type	Returns the type of data that follows the header byte.	
		`0b00`	Reserved.
		`0b01`	A 64-bit or 128-bit data payload follows the header.
		`0b10`	A timestamp value follows the header. **Note** 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` > 64, the payload is zero extended above the 64-bit timestamp value.
		`0b11`	Reserved.

Related references

3.4.2 Timestamp Control register on page 3-44.

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 or 136 bits of data. If the ELA-500 is configured with `GRP_WIDTH` > 64, the timestamp value is zero-extended to 128 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 either 72 bits or 136 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 or 136-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, until all the payload data has been read, that is, two chunks if GRP_WIDTH = 64, four chunks if GRP_WIDTH = 128.

When the final 32 bits of the payload have been read, the RRAR 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.

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

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.

Note

In the following sequence of steps, a lowercase <*n*>, where *n* = 0 to 4, denotes one of the five *Trigger States*.

*Trigger Signal Comparison*s 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>). The *Trigger Condition* is only met if both the *Signal Group* and *External Trigger Input Signal* comparisons succeed.

The mask registers can also support *Trigger Condition*s 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.

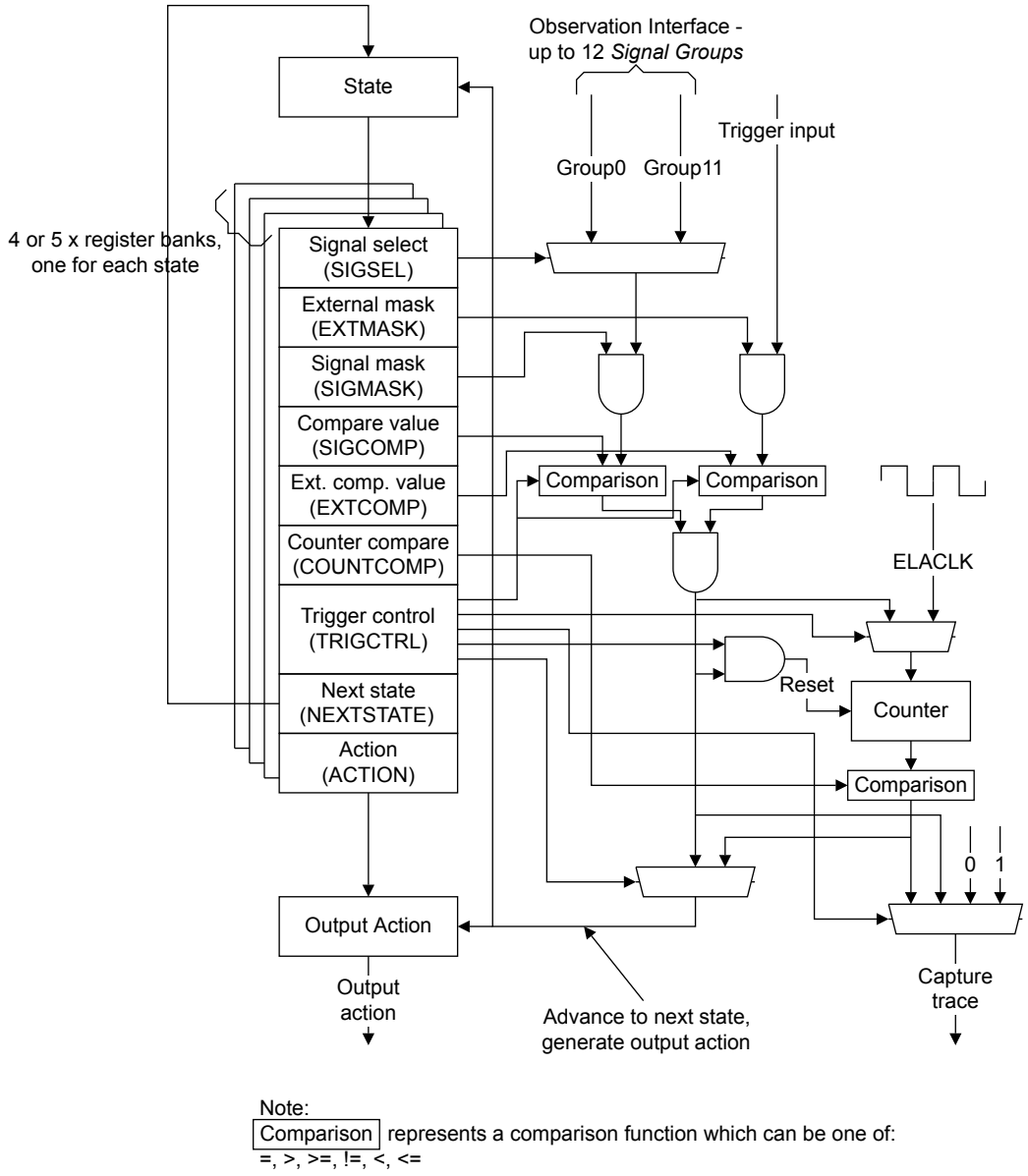

Figure 2-6 Triggering mechanism within the ELA-500

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* 0 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: First condition is programmed for signal comparison, 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: First condition is programmed for signal comparison, 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: First condition is programmed for counter comparison, 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: 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 3-61.

3.10.6 Alt Action registers on page 3-62.

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_CATPURE_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*.

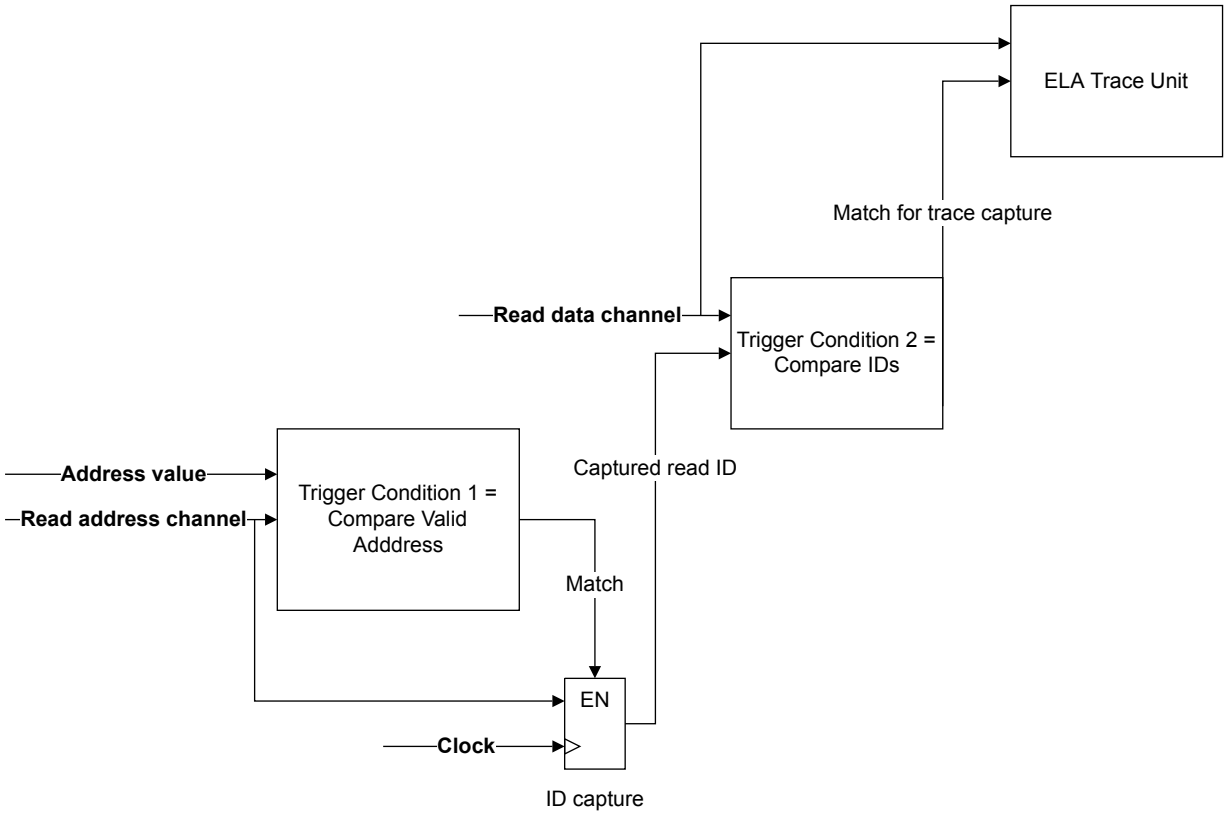

Figure 2-7 Transaction ID capture from read address channel

The parameters `ID_CAPTURE_GEN` = 1 and `COND_TRIG` = 1 must be set to enable generation of the ID capture feature. Also the `ID_CAPTURE_SIZE` parameter must be set to the number of bits in the ID tag.

If DEVID1 [24:20] = 0 then it indicates that the Transaction ID capture feature is disabled, otherwise DEVID1 [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 3-75.

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 automatically set to 0, disabling the ELA-500.

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 eight configurable parameters.

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

Table 2-2 ELA-500 configuration parameters

Parameter	Values	Default	Description
GRP_WIDTH	64 or 128	64	64 or 128 debug signals are allowed in a *Signal Group*.[a]
RAM_ADDR_SIZE	2-30 inclusive.	9	Number of address bits in the SRAM. Number of entries in the SRAM = $2^{RAM\_ADDR\_SIZE}$ For example, when RAM_ADDR_SIZE is 9, the SRAM has 512 entries. ARM recommends at least 512 entries.[b]
TRACE_GEN	0 or 1	1	0: Trace unit not generated.[c]
			1: Trace unit generated.
SECURE_MODE	0 or 1	1	0: Non-secure mode operation. **Note** ARM does not recommend the use of Non-secure mode.
			1: Secure mode operation.
COND_TRIG	0 or 1	0	0: Conditional *Trigger State* feature is disabled.
			1: Conditional *Trigger State* feature is enabled.
ID_CAPTURE_SIZE	2-32	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. **Note** When ID_CAPTURE_GEN=1, you must also set COND_TRIG=1.
NUM_TRIG_STATES	4 or 5	4	Sets the number of *Trigger States*.

[a] When GRP_WIDTH = 64, the APB registers for the extended 128-bit SIGMASK and SICCOMP values are not available.
[b] An entry is data payload plus header byte. When GRP_WIDTH = 64, each entry is 9 bytes. When GRP_WIDTH = 128, each entry is 17 bytes.
[c] When TRACE_GEN == 0, the APB trace registers are not available and the SRAM interface signals are tied to zero.

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:

- *3.20 ID register descriptions* on page 3-78.

3.1 Access permissions

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[d]	Access when CTRL.RUN=0[d]
3.4.1 Logic Analyzer Control register on page 3-44	Can be accessed by software.	Can be accessed by software.
3.5 Current State register summary on page 3-47		
3.13 Software Lock register summary on page 3-69		
3.15 Authentication register summary on page 3-71		
3.17 Device register summary on page 3-73		
3.19 ID register summary on page 3-77		
3.7 RAM register summary on page 3-50	Must not be accessed by software.	
3.11 Integration Mode register summary on page 3-66		
3.9 Trigger State register summary on page 3-54	Can be read by software. Must not be written.	
3.4.2 Timestamp Control register on page 3-44		
3.4.4 Pre-trigger Action register on page 3-45		

[d] 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 <*n*>, 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<n> register.
3. Program the mask and compare registers for each *Trigger State* by writing to SIGMASK<n> and SIGCOMP<n>, respectively.
4. Program the *Output Action* for each *Trigger State* by writing to the ACTION<n> 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<n> 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<n>.
7. Program the Trigger Control register to select the comparison type and counter options for each *Trigger State* by writing to TRIGCTRL<n>.
8. ——— **Note** ———

   Following reset, the RWAR is already set to `0x00000000` and therefore does not need to be explicitly set.

   ———————————

   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 3-39.

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	*3.4.1 Logic Analyzer Control register* on page 3-44
0x004	TIMECTRL	RW	0x00000000[e]	*3.4.2 Timestamp Control register* on page 3-44
0x008	TSSR	RW	0x00000000	*3.4.3 Trigger State Select Register* on page 3-45
0x010	PTACTION	RW	0x00000000	*3.4.4 Pre-trigger Action register* on page 3-45

[e] 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 3-43 provides cross-references to individual registers.

This section contains the following subsections:

- *3.4.1 Logic Analyzer Control register* on page 3-44.
- *3.4.2 Timestamp Control register* on page 3-44.
- *3.4.3 Trigger State Select Register* on page 3-45.
- *3.4.4 Pre-trigger Action register* on page 3-45.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.3 Control register summary* on page 3-43.

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:

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

Configurations Available when `TRACE_GEN` = 1.

Attributes See *3.3 Control register summary* on page 3-43.

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 **ELACLK**. When the specified bit changes, a timestamp packet is requested to be inserted into the trace SRAM when there is an **ELACLK** 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 *Trigger States* 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 3-43.

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** *Trigger State* 4 independent trace disabled. **ALTTS[4]=1** *Trigger State* 4 independent trace enabled. 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 3-43.

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 **ELAOUTPUT[3:0]**.
[3]	TRACE	Enables trace.
[2]	STOPCLOCK	Sets the level to drive on **STOPCLOCK**.
[1:0]	CTTRIGOUT	Sets the value to drive on **CTTRIGOUT[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	*3.6.1 Current Trigger State Register* on page 3-48
0x024	CCVR	RO	0x00000000	*3.6.2 Current Counter Value Register* on page 3-48
0x028	CAVR	RO	0x00	*3.6.3 Current Action Value Register* on page 3-49
0x02C	RDCAPTID	RO	-	*3.6.4 Read Captured Transaction ID register* on page 3-49

3.6 Current State register descriptions

This section describes the ELA-500 Current State registers.

Table 3-7 Current State registers summary on page 3-47 provides cross-references to individual registers.

This section contains the following subsections:

- *3.6.1 Current Trigger State Register* on page 3-48.
- *3.6.2 Current Counter Value Register* on page 3-48.
- *3.6.3 Current Action Value Register* on page 3-49.
- *3.6.4 Read Captured Transaction ID register* on page 3-49.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.5 Current State register summary* on page 3-47.

The following table shows the bit assignments.

Table 3-8 CTSR bit assignments

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

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.5 Current State register summary* on page 3-47.

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 3-47.

The following table shows the bit assignments.

Table 3-10 CAVR bit assignments

Bits	Name	Function
[7:0]	CAVR	Returns the Action value when the CTSR was last read.

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 3-47.

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	-	*3.8.1 RAM Read Address Register* on page 3-51
0x044	RRDR	RO	-	*3.8.2 RAM Read Data Register* on page 3-51
0x048	RWAR	RW	-	*3.8.3 RAM Write Address Register* on page 3-52
0x04C	RWDR	WO	-	*3.8.4 RAM Write Data Register* on page 3-52

3.8 RAM register descriptions

This section describes the ELA-500 RAM registers.

Table 3-12 RAM registers summary on page 3-50 provides cross-references to individual registers.

This section contains the following subsections:

- *3.8.1 RAM Read Address Register* on page 3-51.
- *3.8.2 RAM Read Data Register* on page 3-51.
- *3.8.3 RAM Write Address Register* on page 3-52.
- *3.8.4 RAM Write Data Register* on page 3-52.

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 3-50.

The following table shows the bit assignments.

Table 3-13 RRAR bit assignments

Bits	Name	Function
[`RAM_ADDR_SIZE`-1:0]	RRA	RAM Read Address. Writes to the RRA cause the trace SRAM data at that address to be transferred into the holding register. After the SRAM read data is transferred to the holding register, RRA increments by one. This prepares the RRA address for sequential RRDR reads. 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.

Related references

3.8.2 RAM Read Data Register on page 3-51.

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 3-50.

The following table shows the bit assignments.

Table 3-14 RRDR bit assignments

Bits	Name	Function
[31:0]	RRD	Reads SRAM data from the holding register. 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. 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. The RRA wraps to address zero if it is incremented beyond the maximum depth of the SRAM.

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 3-50.

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 $2^{RAM\_ADDR\_SIZE}$ 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	RAM Write Address. Writes to the RWA set the SRAM address for data that is then written through the RWDR. 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. 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. The RWAR is automatically incremented by APB writes to the SRAM through the RWDR.

Related references

3.8.4 RAM Write Data Register on page 3-52.

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 3-50.

The following table shows the bit assignments.

Table 3-16 RWDR bit assignments

Bits	Name	Function
[31:0]	RWDR	Writes data to the write holding register and initiates an SRAM write when the write holding register is full. Writes to the RWD update the internal write holding register. 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.

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
Trigger State 0 registers				
0x100	SIGSEL0	RW	-	*3.10.1 Signal Select registers* on page 3-57
0x104	TRIGCTRL0	RW	-	*3.10.2 Trigger Control registers* on page 3-58
0x108	NEXTSTATE0	RW	-	*3.10.3 Next State registers* on page 3-60
0x10C	ACTION0	RW	0x00	*3.10.4 Action registers* on page 3-60
0x110	ALTNEXTSTATE0	RW	-	*3.10.5 Alt Next State registers* on page 3-61
0x114	ALTACTION0	RW	0x00	*3.10.6 Alt Action registers* on page 3-62
0x120	COUNTCOMP0	RW	-	*3.10.7 Counter Compare registers* on page 3-63
0x130	EXTMASK0	RW	-	*3.10.8 External Mask registers* on page 3-63
0x134	EXTCOMP0	RW	-	*3.10.9 External Compare registers* on page 3-63
0x140	SIGMASK0[31:0]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x144	SIGMASK0[63:32]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x148	SIGMASK0[95:64]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x14C	SIGMASK0[127:96]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x180	SIGCOMP0[31:0]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x184	SIGCOMP0[63:32]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x188	SIGCOMP0[95:64]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x18C	SIGCOMP0[127:96]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
Trigger State 1 registers				
0x200	SIGSEL1	RW	-	*3.10.1 Signal Select registers* on page 3-57
0x204	TRIGCTRL1	RW	-	*3.10.2 Trigger Control registers* on page 3-58
0x208	NEXTSTATE1	RW	-	*3.10.3 Next State registers* on page 3-60
0x20C	ACTION1	RW	0x00	*3.10.4 Action registers* on page 3-60
0x210	ALTNEXTSTATE1	RW	-	*3.10.5 Alt Next State registers* on page 3-61
0x214	ALTACTION1	RW	0x00	*3.10.6 Alt Action registers* on page 3-62
0x220	COUNTCOMP1	RW	-	*3.10.7 Counter Compare registers* on page 3-63
0x230	EXTMASK1	RW	-	*3.10.8 External Mask registers* on page 3-63

Table 3-17 Trigger state registers summary (continued)

Offset	Name	Type	Reset	Description
0x234	EXTCOMP1	RW	-	*3.10.9 External Compare registers* on page 3-63
0x240	SIGMASK1[31:0]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x244	SIGMASK1[63:32]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x248	SIGMASK1[95:64]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x24C	SIGMASK1[127:96]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x280	SIGCOMP1[31:0]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x284	SIGCOMP1[63:32]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x288	SIGCOMP1[95:64]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x28C	SIGCOMP1[127:96]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
Trigger State 2 registers				
0x300	SIGSEL2	RW	-	*3.10.1 Signal Select registers* on page 3-57
0x304	TRIGCTRL2	RW	-	*3.10.2 Trigger Control registers* on page 3-58
0x308	NEXTSTATE2	RW	-	*3.10.3 Next State registers* on page 3-60
0x30C	ACTION2	RW	0x00	*3.10.4 Action registers* on page 3-60
0x310	ALTNEXTSTATE2	RW	-	*3.10.5 Alt Next State registers* on page 3-61
0x314	ALTACTION2	RW	0x00	*3.10.6 Alt Action registers* on page 3-62
0x320	COUNTCOMP2	RW	-	*3.10.7 Counter Compare registers* on page 3-63
0x330	EXTMASK2	RW	-	*3.10.8 External Mask registers* on page 3-63
0x334	EXTCOMP2	RW	-	*3.10.9 External Compare registers* on page 3-63
0x340	SIGMASK2[31:0]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x344	SIGMASK2[63:32]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x348	SIGMASK2[95:64]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x34C	SIGMASK2[127:96]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x380	SIGCOMP2[31:0]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x384	SIGCOMP2[63:32]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x388	SIGCOMP2[95:64]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x38C	SIGCOMP2[127:96]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
Trigger State 3 registers				
0x400	SIGSEL3	RW	-	*3.10.1 Signal Select registers* on page 3-57
0x404	TRIGCTRL3	RW	-	*3.10.2 Trigger Control registers* on page 3-58
0x408	NEXTSTATE3	RW	-	*3.10.3 Next State registers* on page 3-60
0x40C	ACTION3	RW	0x00	*3.10.4 Action registers* on page 3-60
0x410	ALTNEXTSTATE3	RW	-	*3.10.5 Alt Next State registers* on page 3-61
0x414	ALTACTION3	RW	0x00	*3.10.6 Alt Action registers* on page 3-62
0x420	COUNTCOMP3	RW	-	*3.10.7 Counter Compare registers* on page 3-63

Table 3-17 Trigger state registers summary (continued)

Offset	Name	Type	Reset	Description
0x430	EXTMASK3	RW	-	*3.10.8 External Mask registers* on page 3-63
0x434	EXTCOMP3	RW	-	*3.10.9 External Compare registers* on page 3-63
0x440	SIGMASK3[31:0]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x444	SIGMASK3[63:32]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x448	SIGMASK3[95:64]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x44C	SIGMASK3[127:96]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x480	SIGCOMP3[31:0]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x484	SIGCOMP3[63:32]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x488	SIGCOMP3[95:64]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x48C	SIGCOMP3[127:96]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
Trigger State 4 registers				
0x500	SIGSEL4	RW	-	*3.10.1 Signal Select registers* on page 3-57
0x504	TRIGCTRL4	RW	-	*3.10.2 Trigger Control registers* on page 3-58
0x508	NEXTSTATE4	RW	-	*3.10.3 Next State registers* on page 3-60
0x50C	ACTION4	RW	0x00	*3.10.4 Action registers* on page 3-60
0x510	ALTNEXTSTATE4	RW	-	*3.10.5 Alt Next State registers* on page 3-61
0x514	ALTACTION4	RW	0x00	*3.10.6 Alt Action registers* on page 3-62
0x520	COUNTCOMP4	RW	-	*3.10.7 Counter Compare registers* on page 3-63
0x530	EXTMASK4	RW	-	*3.10.8 External Mask registers* on page 3-63
0x534	EXTCOMP4	RW	-	*3.10.9 External Compare registers* on page 3-63
0x540	SIGMASK4[31:0]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x544	SIGMASK4[63:32]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x548	SIGMASK4[95:64]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x54C	SIGMASK4[127:96]	RW	-	*3.10.10 Signal Mask registers* on page 3-64
0x580	SIGCOMP4[31:0]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x584	SIGCOMP4[63:32]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x588	SIGCOMP4[95:64]	RW	-	*3.10.11 Signal Compare registers* on page 3-65
0x58C	SIGCOMP4[127:96]	RW	-	*3.10.11 Signal Compare registers* on page 3-65

3.10 Trigger State register descriptions

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

Table 3-17 Trigger state registers summary on page 3-54 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 3-57.
- *3.10.2 Trigger Control registers* on page 3-58.
- *3.10.3 Next State registers* on page 3-60.
- *3.10.4 Action registers* on page 3-60.
- *3.10.5 Alt Next State registers* on page 3-61.
- *3.10.6 Alt Action registers* on page 3-62.
- *3.10.7 Counter Compare registers* on page 3-63.
- *3.10.8 External Mask registers* on page 3-63.
- *3.10.9 External Compare registers* on page 3-63.
- *3.10.10 Signal Mask registers* on page 3-64.
- *3.10.11 Signal Compare registers* on page 3-65.

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:

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

Configurations Available in all configurations. SIGSEL<4> only available when `NUM_TRIG_STATES` = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

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>	Selects *Signal Group*.
		`0x1` Selects *Signal Group* 0.
		`0x2` Selects *Signal Group* 1.
		`0x4` Selects *Signal Group* 2.
		`0x8` Selects *Signal Group* 3.
		`0x10` Selects *Signal Group* 4.
		`0x20` Selects *Signal Group* 5.
		`0x40` Selects *Signal Group* 6.
		`0x80` Selects *Signal Group* 7.
		`0x100` Selects *Signal Group* 8.
		`0x200` Selects *Signal Group* 9.
		`0x400` Selects *Signal Group* 10.
		`0x800` Selects *Signal Group* 11.

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. TRIGCTRL<4> only available when `NUM_TRIG_STATES` = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

The following table shows the bit assignments.

Table 3-19 TRIGCTRL<n> register bit assignments

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

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

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

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 *Trigger Signal Comparison* mode.
		0b1	Enable counters and select *Trigger Counter Comparison* mode.
[2:0]	COMP	*Trigger Signal Comparison* type select.	
		0b000	*Trigger Signal Comparison*s disabled. The enabled counters count clocks immediately after the *Trigger State* has been entered and generate a programmable *Output Action* and transition to the next *Trigger State* when the Counter Compare Register count is reached, that is when a *Trigger Counter Comparison* match occurs.
		0b001	Compare type is *equal* (==).
		0b010	Compare type is *greater than* (>).
		0b011	Compare type is *greater than or equal* (>=).
		0b101	Compare type is *not equal* (!=).
		0b110	Compare type is *less than* (<).
		0b111	Compare type is *less than or equal* (<=).

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. NEXTSTATE<4> only available when NUM_TRIG_STATES = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

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 *Trigger Condition* has been met in the current state.	
		0x0	Do not change state. This is the final *Trigger State*.
		0x1	Selects *Trigger State* 0.
		0x2	Selects *Trigger State* 1.
		0x4	Selects *Trigger State* 2.
		0x8	Selects *Trigger State* 3.
		0x10	Selects *Trigger State* 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:

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

Configurations Available in all configurations. ACTION<4> only available when NUM_TRIG_STATES = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

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 **ELAOUTPUT[3:0]**.	
[3]	TRACE	Trace active.	
		0b0	Trace is not active.
		0b1	Trace is active.
[2]	STOPCLOCK	Level to drive on **STOPCLOCK**.	
		0b0	Drive 0 on **STOPCLOCK**.
		0b1	Drive 1 on **STOPCLOCK**.
[1:0]	CTTRIGOUT	Value to drive on **CTTRIGOUT[1:0]**.	

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:

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

Configurations Available when COND_TRIG = 1. ALTNEXTSTATE<4> only available when NUM_TRIG_STATES = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

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>	Selects the next state to move to after the conditional *Trigger Condition* has been met in the current state.	
		0x0	Do not change state. This is the final *Trigger State*.
		0x1	Selects *Trigger State* 0.
		0x2	Selects *Trigger State* 1.
		0x4	Selects *Trigger State* 2.
		0x8	Selects *Trigger State* 3.
		0x10	Selects *Trigger State* 4, when NUM_TRIG_STATES=5.

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. ALTACTION<4> only available when NUM_TRIG_STATES = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

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 **ELAOUTPUT[3:0]**.	
[3]	TRACE	Trace active.	
		0b0	Trace is not active.
		0b1	Trace is active.
[2]	STOPCLOCK	Level to drive on **STOPCLOCK**.	
		0b0	Drive 0 on **STOPCLOCK**.
		0b1	Drive 1 on **STOPCLOCK**.
[1:0]	CTTRIGOUT	Value to drive on **CTTRIGOUT[1:0]**.	

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. COUNTCOMP<4> only available when `NUM_TRIG_STATES` = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

The following table shows the bit assignments.

Table 3-24 COUNTCOMP<n> register bit assignments

Bits	Name	Function
[31:0]	COUNTCOMP<n>	Value that, when reached in the associated up-counter for this *Trigger State*, causes a *Trigger Counter Comparison* 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. EXTMASK<4> only available when `NUM_TRIG_STATES` = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

The following table shows the bit assignments.

Table 3-25 EXTMASK<n> register bit assignments

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

Related concepts

2.5 Triggering on page 2-33.

Related references

3.10.10 Signal Mask registers on page 3-64.

3.10.11 Signal Compare registers on page 3-65.

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:

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

Configurations Available in all configurations. EXTCOMP<4> only available when NUM_TRIG_STATES = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

The following table shows the bit assignments.

Table 3-26 EXTCOMP<n> register bit assignments

Bits	Name	Function
[7:2]	EXTTRIG	Compare value for **EXTTRIG[5:0]** signals.
[1:0]	CTTRIGIN	Compare value for **CTTRIGIN[1:0]** signals.

Related concepts

2.5 Triggering on page 2-33.

Related references

3.10.10 Signal Mask registers on page 3-64.

3.10.11 Signal Compare registers on page 3-65.

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:

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

Configurations Available in all configurations. SIGMASK<4> only available when NUM_TRIG_STATES = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

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.
[127:96]	SIGMASK[127:96]	Mask bits from SIGCOMP[127:96]. These bits are only used if GRP_WIDTH=128.

Related concepts

2.5 Triggering on page 2-33.

Related references

3.10.11 Signal Compare registers on page 3-65.

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. SIGCOMP<4> only available when `NUM_TRIG_STATES` = 5.

Attributes See *3.9 Trigger State register summary* on page 3-54.

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 *Signal Group* signals[31:0].
[63:32]	SIGCOMP[63:32]	Compare value for *Signal Group* signals[63:32].
[95:64]	SIGCOMP[95:64]	Compare value for *Signal Group* signals[95:64]. These bits are only used if `GRP_WIDTH`=128.
[127:96]	SIGCOMP[127:96]	Compare value for *Signal Group* signals[127:96]. These bits are only used if `GRP_WIDTH`=128.

Related concepts

2.5 Triggering on page 2-33.

Related references

3.10.10 Signal Mask registers on page 3-64.

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	*3.12.1 Integration Mode Action Trigger Output register* on page 3-67
0xEF8	ITTRIGIN	RO	-	*3.12.2 Integration Mode External Trigger Input register* on page 3-67
0xF00	ITCTRL	RW	0b0	*3.12.3 Integration Mode Control register* on page 3-68

3.12 Integration Mode register descriptions

This section describes the ELA-500 Integration Mode registers.

Table 3-29 Integration mode registers summary on page 3-66 provides cross-references to individual registers.

This section contains the following subsections:

- *3.12.1 Integration Mode Action Trigger Output register* on page 3-67.
- *3.12.2 Integration Mode External Trigger Input register* on page 3-67.
- *3.12.3 Integration Mode Control register* on page 3-68.

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 3-66.

The following table shows the bit assignments.

Table 3-30 ITTRIGOUT register bit assignments

Bits	Name	Function
[7:4]	ELAOUTPUT	Value to drive on **ELAOUTPUT[3:0]** when ITCTLR.IME = 1.
[3]	Reserved	-
[2]	STOPCLOCK	Level to drive on **STOPCLOCK** when ITCTLR.IME = 1. `0b0` Drive 0 on **STOPCLOCK**. `0b1` Drive 1 on **STOPCLOCK**.
[1:0]	CTTRIGOUT	Value to drive on **CTTRIGOUT[1:0]** when ITCTLR.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 3-66.

The following table shows the bit assignments.

Table 3-31 ITTRIGIN register bit assignments

Bits	Name	Function
[7:2]	EXTTRIG	Captures the value on **EXTTRIG[5:0]** when ITCTLR.IME = 1.
[1:0]	CTTRIGIN	Captures the value on **CTTRIGIN[1:0]** when ITCTLR.IME = 1.

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 3-66.

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 CTRL.RUN = 0.

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	-	*3.14.1 Lock Access Register* on page 3-70
0xFB4	LSR	RO	0b000 or 0b011[f]	*3.14.2 Lock Status Register* on page 3-70

[f] 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 3-69 provides cross-references to individual registers.

This section contains the following subsections:

- *3.14.1 Lock Access Register* on page 3-70.
- *3.14.2 Lock Status Register* on page 3-70.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.13 Software Lock register summary* on page 3-69.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.13 Software Lock register summary* on page 3-69.

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	-	*3.16.1 Authentication Status register* on page 3-72

3.16 Authentication register descriptions

This section describes the ELA-500 Authentication registers.

Table 3-36 Authentication registers summary on page 3-71 provides cross-references to individual registers.

This section contains the following subsections:

- *3.16.1 Authentication Status register* on page 3-72.

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 3-71.

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 2-37.

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.

Table 3-38 Device registers summary

Offset	Name	Type	Reset	Description
0xFBC	DEVARCH	RO	0x47700A75	*3.18.1 Device Architecture register* on page 3-74
0xFC0	DEVID2	RO	-[g]	*3.18.2 Device Configuration register 2* on page 3-74
0xFC4	DEVID1	RO	-[g]	*3.18.3 Device Configuration register 1* on page 3-75
0xFC8	DEVID	RO	-[g]	*3.18.4 Device Configuration register* on page 3-75
0xFCC	DEVTYPE	RO	0x75	*3.18.5 Device Type Identifier register* on page 3-76

[g] Configuration-dependent.

3.18 Device register descriptions

This section describes the ELA-500 Device registers.

Table 3-38 Device registers summary on page 3-73 provides cross-references to individual registers.

This section contains the following subsections:

- *3.18.1 Device Architecture register* on page 3-74.
- *3.18.2 Device Configuration register 2* on page 3-74.
- *3.18.3 Device Configuration register 1* on page 3-75.
- *3.18.4 Device Configuration register* on page 3-75.
- *3.18.5 Device Type Identifier register* on page 3-76.

3.18.1 Device Architecture register

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

The DEVARCH register characteristics are:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.17 Device register summary* on page 3-73.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.17 Device register summary* on page 3-73.

The following table shows the bit assignments.

Table 3-40 DEVID2 bit assignments

Bits	Name	Function
[31:8]	Reserved	
[7:0]	ALTTS	`0x00` NUM_TRIG_STATES=4. *Trigger State* 4 is not implemented. `0x10` NUM_TRIG_STATES=5. *Trigger State* 4 is implemented and can be used for independent trace. All other encodings are reserved and read as `0x00`.

3.18.3 Device Configuration register 1

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

The DEVID1 register characteristics are:

Usage constraints No usage constraints.
Configurations Available in all configurations.
Attributes See *3.17 Device register summary* on page 3-73.

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 *Trigger States*. Four or five.
[15:8]	SIGGRPWIDTH	*Signal Group* width. The field value is (*Signal Group* width/8) - 1. For example, 7 if `GRP_WIDTH` = 64, 15 if `GRP_WIDTH` = 128.
[7:0]	NUMSIGGRPS	Number of *Signal Groups*. 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:

Usage constraints No usage constraints.
Configurations Available in all configurations.
Attributes See *3.17 Device register summary* on page 3-73.

The following table shows the bit assignments.

Table 3-42 DEVID bit assignments

Bits	Name	Function
[31:25]	Reserved	
[24:20]	ID_CAPTURE_SIZE	2-30 bits when `COND_TRIG` = 1, or 0 otherwise.
[19:16]	COND_TRIG	Shows the value of the `COND_TRIG` parameter. 1, when `COND_TRIG` = 1, or 0 otherwise.
[15:8]	SRAM_ADDR_SIZE	SRAM address width in bits.

Table 3-42 DEVID bit assignments (continued)

Bits	Name	Function
[7:4]	TRACEFORMAT	Trace implementation: `1` Fixed at 1. Indicates Trace header format revision 1.
[3:0]	TRACETYPE	ATB trace: `0` ATB trace not implemented. `1` ATB trace is implemented.

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 3-73.

The following table shows the bit assignments.

Table 3-43 DEVTYPE bit assignments

Bits	Name	Function
[7:0]	DEVTYPE	`0x75`. SUB type = `0x7`. MAJOR type = `0x5`.

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	*3.20.1 Peripheral ID4 Register* on page 3-78
0xFD4	PIDR5	RO	0x00	*3.20.2 Peripheral ID5 Register* on page 3-78
0xFD8	PIDR6	RO	0x00	*3.20.3 Peripheral ID6 Register* on page 3-78
0xFDC	PIDR7	RO	0x00	*3.20.4 Peripheral ID7 Register* on page 3-79
0xFE0	PIDR0	RO	0xB8	*3.20.5 Peripheral ID0 Register* on page 3-79
0xFE4	PIDR1	RO	0xB9	*3.20.6 Peripheral ID1 Register* on page 3-79
0xFE8	PIDR2	RO	0x1B	*3.20.7 Peripheral ID2 Register* on page 3-80
0xFEC	PIDR3	RO	0x00	*3.20.8 Peripheral ID3 Register* on page 3-80
0xFF0	CIDR0	RO	0x0D	*3.20.9 Component ID0 Register* on page 3-80
0xFF4	CIDR1	RO	0x90	*3.20.10 Component ID1 Register* on page 3-81
0xFF8	CIDR2	RO	0x05	*3.20.11 Component ID2 Register* on page 3-81
0xFFC	CIDR3	RO	0xB1	*3.20.12 Component ID3 Register* on page 3-81

3.20 ID register descriptions

This section describes the ELA-500 ID registers.

Table 3-44 ID registers summary on page 3-77 provides cross-references to individual registers.

This section contains the following subsections:

3.20.1 Peripheral ID4 Register

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

The PIDR4 characteristics are:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.19 ID register summary* on page 3-77.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.19 ID register summary* on page 3-77.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.19 ID register summary* on page 3-77.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.19 ID register summary* on page 3-77.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.19 ID register summary* on page 3-77.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.19 ID register summary* on page 3-77.

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 3-77.

The following table shows the bit assignments.

Table 3-51 PIDR2 bit assignments

Bits	Name	Function
[7:4]	REVISION	`0x1`. Revision number. Indicates revision r1p0.
[3]	JEDEC	`0x1`. 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 3-77.

The following table shows the bit assignments.

Table 3-52 PIDR3 bit assignments

Bits	Name	Function
[7:4]	REVAND	`0x00`. RevAnd.
[3:0]	CMOD	`0x00`. Indicates whether the customer has modified the behavior of the component. In most cases, this field is 0b0000. Customers change this value when they 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 3-77.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.19 ID register summary* on page 3-77.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.19 ID register summary* on page 3-77.

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:

Usage constraints No usage constraints.

Configurations Available in all configurations.

Attributes See *3.19 ID register summary* on page 3-77.

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 in its full configuration.

It contains the following sections:

A.1 Clocks and reset

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

Table A-1 ELA-500 clock and reset signals

Signal name	Type	Description
ELACLK	Input	Logic analyzer clock for triggering and trace.
RESETn		Reset for **ELACLK** domain including *Output Actions*.
PCLKDBG		Clock for Debug APB interface.
PRESETDBGn		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
PSELDBG	Input	Select.	Connect to the CoreSight Debug subsystem.
PENABLEDBG		Enable.	
PWRITEDBG		Peripheral write.	
PADDRDBG[11:2]		Address. The ELA-500 only supports word-aligned addresses.	
PADDRDBG31		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.	
PWDATADBG[31:0]		Write data.	
PREADYDBG	Output	Peripheral ready.	
PSLVERRDBG		Slave error.	
PRDATADBG[31:0]		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.

Table A-3 ELA-500 Observation interface signals

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

[h] 64 or 128 signals, depending on `GRP_WIDTH`.

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
DBGEN	Input	Invasive debug enable.	From CoreSight debug subsystem authentication control signals.
NIDEN		Non-invasive debug enable.	
SPIDEN		Secure invasive debug enable.	
SPNIDEN		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
DFTRAMHOLD	Input	Disables the RAM chip select during scan shift.	Connect to scan DFT logic.
MBISTREQ		MBIST Mode Request. Enables MBIST testing.	Connect to MBIST controller.
nMBISTRESET		Resets functional logic to enable MBIST operations.	
MBISTADDR[RAM_ADDR_SIZE-1:0]		Logical RAM address.	
MBISTINDATA[(GRP_WIDTH+8)-1:0]		RAM Write data.	
MBISTWRITEEN		Write enable control. A no-op occurs if write and read enables are both zero.	
MBISTREADEN		Read enable control. A no-op occurs if write and read enables are both zero.	
MBISTACK	Output	MBIST Mode Ready. The ELA-500 acknowledges that it is MBIST-ready.	
MBISTOUTDATA[(GRP_WIDTH+8)-1:0]		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.

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

Signal name	Type	Description	Connection information
ELAQREQn	Input	Quiescence request from the clock controller to the ELA-500.	Connect to clock controller or power controller.
ELAQACCEPTn	Output	Quiescence request accept from the ELA-500.	
ELAQDENY		Quiescence request deny from the ELA-500.	
ELAQACTIVE		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 IO, 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
CTTRIGIN[1:0]	Input	Trigger inputs.	From external CTI or CoreSight EVENT interface.
EXTTRIG[5:0]		External inputs for non-debug signal *Trigger Condition*.	From other ELA-500 **ELAOUTPUT** signals or user-defined synchronized signals.

Appendix B
Revisions

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

It contains the following sections:

B.1 Revisions

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.	*3.3 Control register summary* on page 3-43	0000_02
Usage constraints updated.	*3.4.2 Timestamp Control register* on page 3-44	0000_02
	3.10.1 Signal Select registers on page 3-57	0000_02
	3.10.2 Trigger Control registers on page 3-58	0000_02
	3.10.3 Next State registers on page 3-60	0000_02
	3.10.4 Action registers on page 3-60	0000_02
	3.10.7 Counter Compare registers on page 3-63	0000_02
	3.10.8 External Mask registers on page 3-63	0000_02
	3.10.9 External Compare registers on page 3-63	0000_02
	3.10.10 Signal Mask registers on page 3-64	0000_02
	3.10.11 Signal Compare registers on page 3-65	0000_02
	3.4.4 Pre-trigger Action register on page 3-45	0000_02
Reset value for ACTION0, ACTION1, ACTION2, and ACTION3 registers updated.	*3.9 Trigger State register summary* on page 3-54	0000_02
Minor editorial updates and corrections.	Throughout the document	0000_02

Table B-3 Differences between Issue 0000_02 and Issue 0100_00

Change	Location	Affects
Addition of r1p0 parameters table.	*2.7 Parameter summary* on page 2-38	0100_00
Addition of DEVID2 register.	*3.18.2 Device Configuration register 2* on page 3-74	0100_00
Addition of FINALSTATE bit in CTSR.	*3.6.1 Current Trigger State Register* on page 3-48	0100_00
DEVID.TRACEFORMAT fixed at 1.	*3.20.7 Peripheral ID2 Register* on page 3-80	0100_00
PIDR2 revision ID changed to 1.Version r1p0 implements Trace header format revision 1 only.	*3.18.4 Device Configuration register* on page 3-75	0100_00
Addition of *Trigger Signal Alternative Comparison* state.	*1.2 Definitions of terms used in this book* on page 1-13 and throughout the document	0100_00
Addition of *Trigger State* 4 .	Throughout the document	0100_00
Addition of Transaction ID capture.	Throughout the document	0100_00