arm

Using the CoreSight ELA-500 Embedded Logic Analyzer with Arm DS-5

Version 1.0

Non-Confidential

Copyright $\ensuremath{\mathbb{C}}$ 2020 Arm Limited (or its affiliates). All rights reserved.

Issue 01 102623_0100_01_en



Using the CoreSight ELA-500 Embedded Logic Analyzer with Arm DS-5

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	1 January 2020	Non-Confidential	First release

Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

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

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

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or [™] are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at https://www.arm.com/company/policies/trademarks.

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

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.

Feedback

Arm[®] welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/ documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction	6
2. Before you begin	8
3. Importing the CoreSight ELA-500 DTSL Use case scripts	9
4. Configuring the CoreSight ELA-500 DTSL Use case scripts	13
5. Running the DS-5 ELA use case scripts	
6. Capturing the ELA trace data	19
7. Analyzing the ELA trace capture	21

1. Introduction

The Arm CoreSight ELA-500 Embedded Logic Analyzer provides low level signal visibility into Arm IP and third party IP. When used with a processor, it provides visibility of load, stores, speculative fetches, cache activity, and transaction life cycle, none of which are available through instruction tracing.

CoreSight ELA-500 enables swift hardware assisted debug of otherwise hard-to-trace issues, including data corruption and dead/live locks. As well as accelerating debug cycles during complex IP bring up, it provides extra assistance for post deployment debug.

CoreSight ELA-500 offers on-chip visibility of both Arm and proprietary IP blocks. Trigger conditions can be programmed over standard debug interfaces either directly by an on-chip processor or an external debugger.

The Problem

One of the most common deadlock scenarios can be caused when a processor initiates memory transactions to a location in the system in which no bus slave exists or the bus slave has limitations such as not being able to handle burst transactions. This type of incomplete transaction can ultimately lead to the processor locking-up (deadlock).

In a perfect world, systems should be designed in such a way that all the entire physical memory map is fully populated. This means that all memory transactions, to all addresses, will correctly respond with either a valid transaction result or a bus fault. However, for certain designs this may not always be the case. The aggressive speculation and prefetching performed by Arm processors mean that these memory map "holes" are more likely to be exposed by incorrect software, even if these memory "holes" are not explicitly referenced by software.

Software can prevent this by correctly configuring the MMU translation tables to accurately describe the physical memory map. Software should configure any memory map "holes" as being Invalid. Configuring the MMU this way will prevent the processor from making any physical bus transactions to that location, and ultimately preventing this type of deadlock scenario.

Debugging these types of deadlock scenarios pose an issue when debugging using traditional methods, such as external debug, and instruction / data trace. A processor core which has locked-up due to an incomplete transaction, might not be able to enter halt mode debug. Effectively, the external debugger is unable to break the processor and inspect its internal state. Trace capture might still be available, but it will not provide any record of the speculative or prefetched transactions that could be responsible for the deadlock.

The Solution

The CoreSight ELA-500 can be used in this scenario to trace the external bus transactions made by the processor (both explicitly and speculatively). This tutorial intends to show the use case scripting capabilities of DS-5, and demonstrate the example CoreSight ELA-500 use case script shipped with Arm DS-5 Development Studio.

About the CoreSight ELA-500

The ELA-500 can be implemented with up to 12 Signal Groups, each containing 64, 128, or 256 signals. The connections between the signals in the signal groups is dependent on the system and the IP that it is connected to. The specific signal interfaces is documented in the relevant documentation (low-level signal description documents like this are typically not publicly available, and are made available only to licensees of the Arm IP). Arm IP connected to an ELA is supplied with a JSON file which documents and annotates the signal group connections for that particular IP, in a machine-readable format. The JSON file can be interpreted by DS-5 to allow seamless debugging of a piece of IP using DS-5 and the ELA.

Signals typically consist of debug signals (status or output), and qualifiers (trigger). Qualifier signals might be required to determine that the debug signal is valid. Debug signals are valid when the qualifier signal(s) are asserted.

The System

For the purposes of this tutorial, the Cortex-A72 + ELA-500 system utilizes the LAK-500A. The LAK-500A is an Integration Kit for the ELA-500, and the Cortex-A72, it is an add-on to the ELA-500. The LAK-500A exposes a number of pre-defined debug observation ports to the Cortex-A72 (Signal Groups), and provides the corresponding JSON signal mapping file.

As part of the LAK-500A, one of the debug observation ports to the Cortex-A72 exposes the physical read address signal bus ARADDR and an address valid signal, ARVALID.

These signal names have been obfuscated for this tutorial.

These signals are required to determine the read addresses issued by the core, prior to the "lockup". Post analysis of these read transactions will help identify which transaction may have caused the fault.

2. Before you begin

Make sure you have installed DS-5 v5.26 or later.

The scripts that enable DS-5 to work with the ELA were added to DS-5 v5.25. However the Use case scripts import method that we refer to below, was added in DS-5 v5.26.

3. Importing the CoreSight ELA-500 DTSL Use case scripts

With the following steps, you can import the CoreSight ELA-500 DTSL Use case scripts:

- 1. Launch Eclipse for DS-5 from the Start Menu.
- 2. Select a Workspace for your DS-5 projects. The default workspace is fine.
- 3. Close the Welcome screen, if it appears.
- 4. Select File > Import... to open the Import Selection dialog.
- 5. Expand the DS-5 group and select Examples and Programming Libraries, click Next.

Figure 3-1: Import Examples and Programming Libraries

Import	
Select	
Imports the DS-5 examples to current workspace	
Select an import wizard:	
✓ 🦢 C/C++	^
C/C++ Executable	
C/C++ Project Settings	
Existing Code as Makefile Project	
> 🗁 CVS	
✓ ▷ DS-5	
C Examples & Programming Libraries	
> 🗁 Install	
> Even / Debug	
Scatter File Editor	
Generation Editor	
	~
? < Back Next > Finish	Cancel

- 6. Expand the **Examples** group, then expand the **Debug and Trace Services Layer (DTSL)** group.
- 7. Select DTSLELA-500.

Figure 3-2: DTSLELA-500 check-box under Debug and Trace Services Layer (DTSL)

Import Examples and Programming Libraries		×
Import DS-5 Examples and Programming Libraries		
Some projects cannot be imported because they already exist in the workspace or in workspace directory		
type filter text]
Examples Armv7 Bare-Metal Armv8 Bare-Metal Bare-Metal Boards CMSIS-RTOS RTX v4 for Cortex-A/R CoreSight Access Library Debug and Trace Services Layer (DTSL) DTSLELA-500 DTSLExampleConfigdb DTSLJavaExample DTSLJavaExample DTSLPythonExample DTSLShowDeviceRegisters DTSLSTMDecode DTSLSTMFileDecodeExample DTSLTraceDume		
Working sets Add project to working sets Working sets:	New Select	
Copyright © 2020 Arm Limited (or its affiliates). All rights reserved. Back Non-Confidential	Cance	l 11 of 22

8. Click Finish.

Result: The **Project Explorer** view populates with the project.

4. Configuring the CoreSight ELA-500 DTSL Use case scripts

To configure the ELA-500, you can either edit a use case script or use the configuration GUI interface. The application specific use case script allows you to script a specific debug recipe. The debug recipe is used to debug a specific debug scenario with the ELA-500. An example of this can be found by navigating to the following use case script:

Scripts window > Use case > DTSLELA-500 > ela_example.py > Configure ELA

For this demonstration, we will use the GUI ELA-500 Configuration Utility to configure the ELA-500 for our specific debug scenario.

DS-5 must be connected to the target SoC before starting configuration.

- 1. Connect to the target.
- 2. Open the GUI ELA-500 configuration utility:
 - a. Navigate to: Scripts window > Use case > DTSLELA-500 > ela_lowlevel.py > Configure ELA
 - b. Right click Configure ELA and select Configure

Figure 4-1: Configure option under the scripts menu

The LEA Long and Paper Report Network	C DS-5 Debug - Edipse Platform							σx
Image: International internatione internatinternatedinternational international international int	File Edit Navigate Search Project Run Window Help							
0 Debug Conter il Dirage Labora statuto 	0 * 0 5 0 8 0 * 19 * 19 * 10 * 9 0 * 0 *					Qu	ick Access	e 1 1 1 1 1 1 1 1
State:::::::::::::::::::::::::::::::::::	● Debug Control III Schelpts Explorer of Energie System = 0 ■ B \ \ \ \ \ B \ \ \ \ B \ \ \ C \ \ B \ \ A \ C \ I \ \ B \ \ A \ C \ I \ \ C \ C \ I \ C \ C \ I \ C \ C \ I \ C \ C	Commands # History & Sorgs 1 Commands # History & Sorgs 1 Commands # History & Sorgs 1 Commands # History & History & History & History & History & Commands # History & Commands # History & Commands # History & Commands # History & Histor	-500. configured.	+=+, +, +, +, +, +, +, +, +, +, +, +, +, +	Hi YunL, № Reg. Hi Reg. Register Set: All registers Name Vale * © AArdol 773 0 * © AArdol 2787 0	B Inf Day, or Func, or Select TO A down* Strakent FLA down* (753 registers (767 registers)	e Access	
 R Datasently (Memory R) = Stack & have = Models = (Levers B) Codes Addens Stack Color Addens Start Color Addens Start Color Addens Constraints <l< td=""><td>Status: connected</td><td>* TIMECTRL: Timestamp Control * TSSR: Trioger State Select</td><td></td><td>~</td><td></td><td></td><td></td><td></td></l<>	Status: connected	* TIMECTRL: Timestamp Control * TSSR: Trioger State Select		~				
Indexes Set Out Interview Address Data (Maxadecina): 4 Systes) Address Interview Interview Interview Interview Interview Address Interview Interview Interview Interview Interview Address Interview Interview Interview Interview Interview An optional size may be entered in sither field. These magnetions will be revealuated each time the view spokines. Some expressions may only be endered in aither field. These magnetions will be revealuated each time the view spokines. Some expressions may only be endered in aither field. These magnetics in the Stack Pointer "Some" "Start" Object memory consently referenced from the Stack Pointer "Some" Some "Start" Object memory consently referenced from the Stack Pointer "Some" Some "Start" Object memory consently referenced from the Stack Pointer "Some" Some "Some S" - Object memory consently referenced from the Stack Pointer "Some" Some "Some S" - Object memory constraining colors plan 5 bytes Consent Asite Some Some Start Consent of memory constraining colors plan 5 bytes Some consent of memory constraining colors plan 5 bytes Some Start Consent A			Disassembly E Memory #	Stack # Trace # Modules II Eve	ents 🕽 Outline	H *	4 = 5. = 4	/ 0
Control of the second sec			R .	\$10	nket ELA demo			-
Address: Inter an address to read from or solect one from Natory. See: Part optional size may be entered as the number of bytes to read, defaults to 1004 bytes. Expression: An optional size may be entered in aither faid, These approxibions will be reevaluated each time the view updates. Some expressions may only be enduated when the target's not number of the enduated when the target's number of the enduated when the target's number of target's number of the enduated when the target's number of target's number of the enduated when the target's number of target's number of the enduated when the target's number of target's number of target's number of target's number of target's number of targ			Address	Data /	Mayadacimal: A hutas			
			Address: Enter an address to read Size: An optional size may be or only be outside address of the outside address of the outside address "SP" - Display memory or "toolses(S)" - Display memory "toolses(S)" - Display memory "toolses(S)" - Display memory "toolses(S)" - Display memory "toolses(S)" - Display memory toolses(S)" - Display memory "toolses(S)" - Display memory toolses(S)" - Display	tom or select one from Natory. extend as the number of bytes to re- ineed in other number of bytes to re- length in not number tanged in not number contents of memory containing colo- tantian of memory containing col- dimension of memory containing col- dimension of memory containing col- angers (Dil+Space) for Content Au	ad, defaults to 1024 bytes, ns will be reevaluated each to sinter ex[0] es plus 5 bytes sist.	ne the view updates. So	ime expression	smay
			<					2 C

Copyright $\ensuremath{\mathbb{C}}$ 2020 Arm Limited (or its affiliates). All rights reserved. Non-Confidential

- 3. Configure the common controls:
 - a. Under the **Common** tab, in the **Pre-trigger action** section, select **Enable trace**.

This configures the ELA to start tracing when it is enabled - it sets **PTACTION.TRACE** so that trace becomes active when the ELA-500 is enabled. When trace is active, trace capture can be controlled to capture on either each ELA clock cycle, a trigger Signal Comparison match, or a trigger Counter Comparison match.

Figure 4-2: Enable trace check-box under the Common tab

Use case script config	guration						×
Use case script confi	guration						
ela_lowievel.py - Config	ure ELA						
ቀ 🗎 X 2a 2d	Name of configuration:	default					
default	Common Trigger Stat	e 0 Trigger St	ate 1 Trigger	State 2 Trigger Sta	ite 3 Trig	gger State	4
	ELA-500 device name	CSELA					
	✓ Timestamp Enable						
	Timestamp Interval	0					
	Trace Counter 0 select	0					
	Trace Counter 1 select	0					
	Trigger State 4 indep	endent trace					
	Pre-trigger action						
	Value to drive on CTTR	IGOUT[1:0]	0x0				
	✓ Value to drive on S	TOPCLOCK					
	Value to drive on ELAC	OUTPUT[3:0]	0x0				
				Apply		Revert	
?				ОК		Cancel	

b. Click Apply

- 4. We now need to configure our initial trigger:
 - a. Open the **Trigger State 0** tab.

Figure 4-3: Trigger State 0 tab

🗢 Use case script config	guration	- D X
Use case script confi	guration	
ela_lowlevel.py - Config	ure ELA	
💠 🗈 🗶 🗠 🗹	Name of configuration: default	
derdan	Common Trigger State 0 Trigger State 1 Trigg	er State 2 Trigger State 3 Trigger State 4
	Select Signal Group 0x1	^
	Trigger Control	
	Signal Comparison (COMP)	Equal
	Comparison mode (COMPSEL)	0
	Counter reset (WATCHRST)	0
	Counter source (COUNTSRC)	0
	Trace capture (TRACE)	0
	Counter clear (COUNTCLR)	0
	Loop counter break (COUNTBRK)	0
	Use of captured ID (CAPTID)	0
	Alternative Signal Comparison (ALTCOMP)	Disabled ~
	Alternative Comparison mode (ALTCOMPSEL)	0
	Novt State Ox1	v
		Apply Revert
?		OK Cancel

b. Set Select Signal Group value to 0x1.

This selects the Signal Group we want to trigger on, which includes the qualifier signal(s), and sets $s_{IGSEL0} = 0x1$. This means that Trigger State 0 will be associated with the trigger signals in Signal Group 0. The ELA-500 uses a 'ones hot' encoding for the Signal Group in the Signal Select registers. In our example, Cortex-72 + ELA-500 + LAK-500A, the RVALID signal resides in Signal Group 0. To locate the RVALID signal location for other targets, check your IP's corresponding JSON file or documentation.

c. Set Signal Comparison (COMP) to Equal.

This sets the Signal Comparison condition. In this case, we want to trigger when the ARVALID signal is valid (ACTIVE HIGH).

d. Set the Next state value to 0×1 .

Here we set the Next state. This is the ELA state we will enter when we meet the trigger condition. In our case we want to capture on each ARVALID assertion, which uses the 'ones hot' for Trigger state 0.

e. Set both the Signal Mask [95:64] and Signal Compare [95:64] fields to 0x00080000.

We need to set Trigger State O's Signal Compare and Signal Mask values for Signal Group O to monitor the ARVALID signal. The bit position of the ARVALID signal is documented in your IPs corresponding JSON file or documentation.

In our example, ARVALID is mapped to bit 83, so we need to specify 0x00080000 in both the Signal Mask [95:64] and Signal Compare [95:64] fields.

Figure 4-4: Signal Mask and Signal Compare fields set to 0x00080000

Use case script configure	uration	— 🗆	×
Use case script config	uration		
ela_lowlevel.py - Configu	ire ELA		
🜩 🗈 🗙 🔤 💋 default	Name of con	figuration: default gger State 0 Trigger State 1 Trigger State 2 Trigger State 3 *1	
	EXTIRIG[5:0		
	[31:0]	0x0	
	[63:32]	0x0	
	[95:64]	0x00080000	
	[127:96]		
	[191:160]	0x0	
	[223:192]	0x0	
	[255:224]	0x0	
	Signal Com	pare	
	[31:0]	0x0	
	[95:64]	0x00080000	
	[127:96]	0x0	~
		Apply Revert	
0		OK Cancel	

f. Click Apply > OK.

5. Running the DS-5 ELA use case scripts

Running the DS-5 ELA use case scripts:

- 1. Program the ELA configuration registers:
 - a. Navigate to: Scripts window > Use case > DTSLELA-500 > ela_lowlevel.py > Configure ELA
 - b. Right-click Configure ELA and select Run...
- 2. Run the ELA:
 - a. Navigate to: Scripts window > Use case > DTSLELA-500 > ela_control.py > Run ELA-500
 - b. Right click Run ELA-500 and select Run...
- 3. Run the target.

Result: The target will run and the ELA will be monitoring the input Signal Group for the trigger conditions.

6. Capturing the ELA trace data

Capturing the ELA trace data:

- 1. The core is unable to enter halt mode debug in our debug scenario, so we need to stop the ELA:
 - a. Navigate to Scripts window > Use case > DTSLELA-500 > ela_control.py > Stop ELA-500
 - b. Right-click **Stop ELA-500** and select **Run...**
- 2. Dump and decode the ELA trace:
 - a. Navigate to: Scripts window > Use case > DTSLELA-500 > ela_example.py > Decode trace data

The **Decode trace data** script requires the corresponding JSON file to be named example_ela_connection.json. The location of this file can be found in the **DTSLELA-500** directory.

- b. Right click Decode trace data and select Configure
- c. Check that Signal group 0 is selected for State 0 and click OK

Figure 6-1: Signal group 0 selected for State 0

♦ Use case script configuration					×		
Use case script config ela_example.py - Decode	uration e trace data						
🕈 🗎 🗙 远 🖄 default	Name of	configuration:	default				
	ELA-500	device name	CSELA				
	Signal g	0					
	State 1	None				~	
	State 2	None				~	1
	State 3	None				×.	
	State 4	None				×.	
				Apply		Revert	
?				ОК		Cancel	

d. Right click **Decode trace data** and select **Run...**

7. Analyzing the ELA trace capture

As a result of our work above, the ELA traced each read transaction and stored them into a circular buffer. The circular buffer holds x number of read transactions, where x relates to the size of the ELA-500 SRAM and number of signals. The read transactions were generated by both explicit reads and speculative reads. You can identify rogue accesses to the potential holes in the memory map by analyzing the read transactions.

The snippet of our trace capture below shows there were several accesses, explicitly called, which were outside the bounds of the memory copy routine. The last address explicitly read by the core was 0x01001fc0. The processor prefetcher continued to read memory from 0x01002000, 0x01002040 and 0x01002080. These memory accesses were to addresses that were outside the internal SRAM. To fix this, we would configure these addresses in the translation tables as Invalid, to prevent the prefetcher from prefetching from this region of memory.

Address read valid	= 0x1
Shareability	= Inner Shareable
Execution state	= AARCH64
Cache Attr	= Write-back, read/write allocate
Access size	= 64 bytes
Read address	= 0x01001fc0
Address read valid	= 0x1
Sharability	= Inner Shareable
Execution state	= AARCH64
Cache Attr	= Write-back, read/write allocate
Access size	= 64 bytes
Read address	$= 0 \times 01002000$
Address read valid	= 0x1
Shareability	= Inner Shareable
Execution state	= AARCH64
Cache Attr	= Write-back, read/write allocate
Access size	= 64 bytes
Read address	$= 0 \times 01002040$
Address read valid	= 0x1
Shareability	= Inner Shareable
Execution state	= AARCH64

Cache Attr	= Write-back, read/write allocate
Access size	= 64 bytes
Read address	= 0x01002080