Arm Debugger Tutorial

Revision: NA

Using the ELA-600 with Arm DS

Non-Confidential

Issue 0.0

NA

Arm Debugger Tutorial

Using the ELA-600 with Arm DS

Release information

Document history

Issue	Date	Confidentiality	Change
0.0	29th of April 2021	Non-Confidential	First version

Confidential Proprietary Notice

Confidentiality Status

This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Product Status

The information in this document is release quality.

Web Address

http://www.arm.com

Contents

1 Introduction

The Arm CoreSight ELA-600 Embedded Logic Analyzer provides low-level signal visibility into Arm IP and third-party IP. When used with a processor, it provides visibility of the following:

- Loads
- Stores
- Speculative fetches
- Cache activity
- Transaction life cycle.

CoreSight ELA-600 offers on-chip visibility of both Arm and proprietary IP blocks. Program Trigger conditions over standard debug interfaces either directly by an on-chip processor or an external debugger.

CoreSight ELA-600 enables swift hardware-assisted debug of otherwise hard-to-trace issues, including data corruption and dead or live locks. It also accelerates debug cycles during complex IP bring up and provides extra assistance for post deployment debug.

CoreSight ELA-600 provides a superset of the functionality of the ELA-500. The largest functional additions are acquiring trace data over the Advanced Trace Bus (ATB) and having potentially eight Trigger States instead of five.

Find more documented information about the ELA-600 in the Resources section of this tutorial.

1.1. The problem

Memory data corruption occurs when on-chip operations like unintended code execution or stack or heap leakage or externally like through malicious code. So, it is useful to have a way to monitor exactly what memory transactions are occurring for certain regions of memory and act accordingly.

In this tutorial scenario:

- Memory accesses to a specific address (`0xB1000000`) are tracked three times.
- The ELA-600 monitors the memory accesses and wait for any memory corruption to occur.
- When memory corruption occurs, the ELA-600 sends a halt to the core through the Cross Trigger Interface (CTI).
- The ELA-600 counts how many cycles occur between sending the halt request and the core sending a halt acknowledgment.

This scenario is modeled on the Data Corruption Scenario found in the Application Note - Arm CoreSight ELA-600 Version 1.0.

1.2. The solution

CoreSight ELA-600 is used in this scenario to trace the external bus transactions made by the processor. This tutorial shows the use case scripting capabilities of Arm Development Studio (Arm DS). It also demonstrates the example CoreSight ELA-600 DTSL use case scripts shipped with Arm DS.

1.3. About the CoreSight ELA-600

ELA-600 implementations can have up to 12 Signal Groups. Each Signal Group can contain 64, 128, or 256 signals. The connections between the signals in the Signal Groups depend on the system and the IP that it is connected to.

The specific signal interfaces are documented in the relevant target or implementation documentation. Low-level signal description documents are typically not publicly available. These documents are only made available to licensees of the Arm IP.

Arm IP connected to an ELA is supplied with a JSON file. The JSON file documents and annotates the Signal Group connections for that particular IP, in a machine-readable format. Arm DS interprets the JSON file to allow seamless debugging of the IP using Arm DS and the ELA. An example of a JSON file can be found in the Arm DS ELA-600 deliverables, *<Arm DS installation directory>\examples\DTSL_examples.zip\DTSLELA-600\axi_interconnect_mapping.json*.

Signals typically consist of debug signals like status or output and qualifiers like triggers. Use qualifier signals to determine if the debug signal is valid. Debug signals are valid when the qualifier signal or signals are asserted.

1.4. The system

For the purposes of this tutorial, we use an example Cortex-A55 + ELA-600 + CCI-500 system. The system exposes several pre-defined debug observation ports known as Signal Groups. Arm DS provides the system JSON signal-mapping file.

CCI-500 Partition P1 is connected to Signal Group 0 of the ELA-600. The following are the CCI-500 signals of interest in this case:

- `VALID_P1` at Signal Group 0 bit position 127.
- `Address_P1` at Signal Group 0 bit position 114:75.
- `Type_P1` at Signal Group 0 bit position 73.

These signals are required to determine the accesses issued by the core to locate the memory corruption. Post analysis of these read transactions allows tracking of:

1. Three of the memory space transactions. The last transaction is the memory corruption.

2. The counter value between the halt request and the halt request acknowledgment.

2 Before you begin

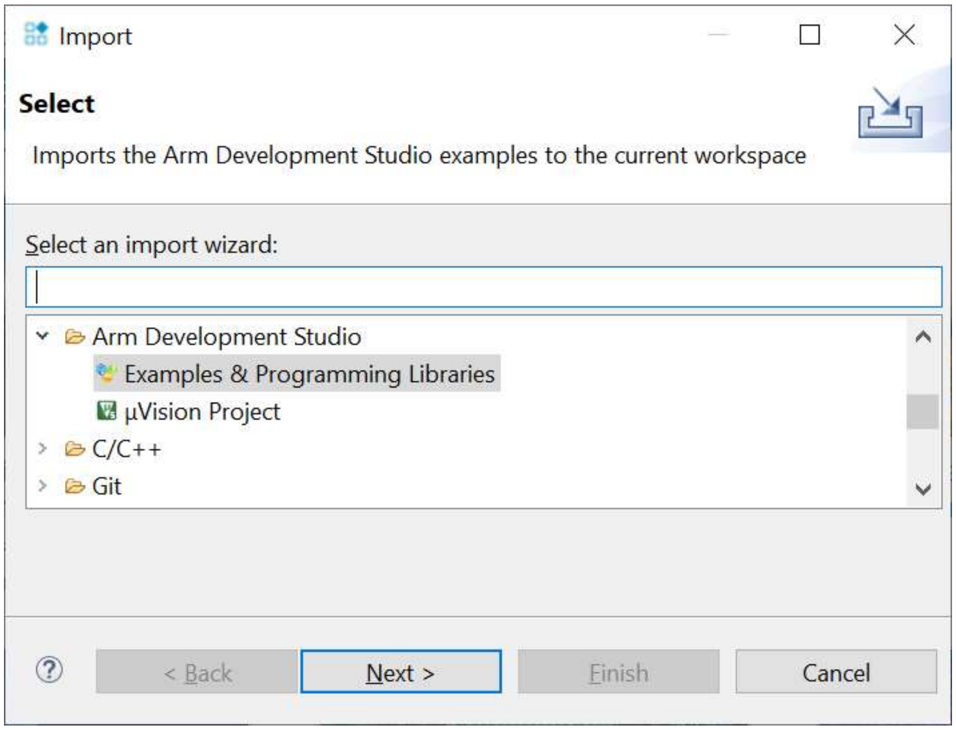
If you have not already, install Arm Development Studio.

Before you begin this tutorial, you must have the following:

- A platform configuration that:
    - Lists the relevant ELA trace component or components.
    - If ELA-600 trace is captured to the ATB interface, lists the component connections and their mapping between the ELA and its trace sink.
    - If applicable, lists the CTIs and their connections.
- A JSON file that contains a list of the components of your IP and their corresponding Signal Group connections. This file is available from the IP designer.

3 Importing the DTSL ELA-600 use case scripts

1. Launch Arm DS IDE.
2. If prompted, select a Workspace for your Arm DS projects. The default workspace is fine.
3. Select File > Import... .
4. Expand the Arm Development Studio and select Examples & Programming Libraries, click Next.


Import

Select

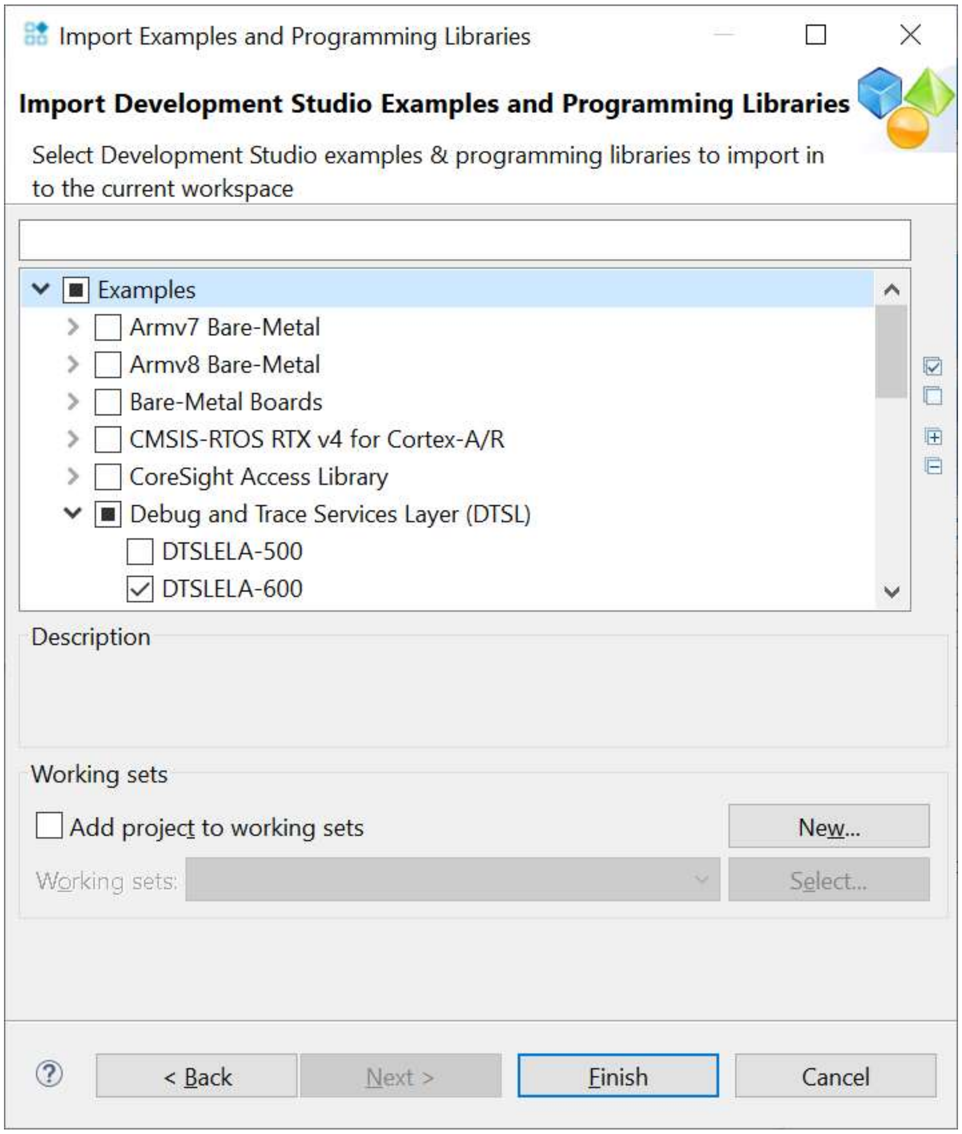
Imports the Arm Development Studio examples to the current workspace

Select an import wizard:

- Arm Development Studio
  - Examples & Programming Libraries
  - µVision Project
- C/C++
- Git

5. Expand Examples > Debug and Trace Services Layer (DTSL).

6. Select DTSLELA-600.


Import Examples and Programming Libraries
Import Development Studio Examples and Programming Libraries
Select Development Studio examples & programming libraries to import in to the current workspace
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
DTSLELA-600
Description
Working sets
Add project to working sets
New...
Working sets:
Select...
< Back
Next >
Finish
Cancel

7. Click Finish.

Result: In the Project Explorer view, the DTSLELA-600 project appears.

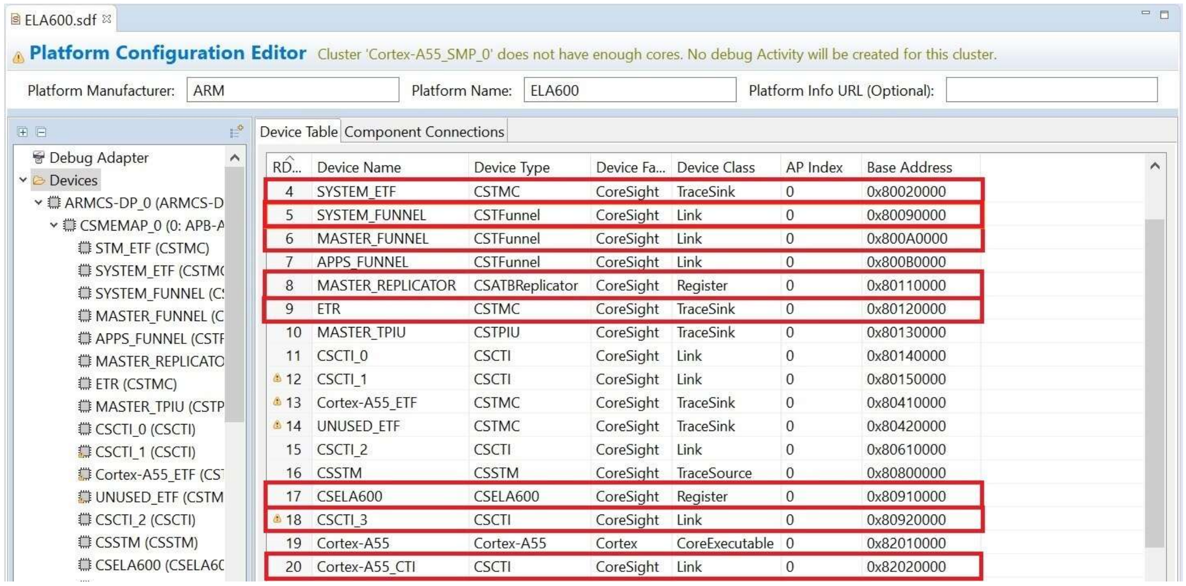
4 Getting an ELA platform configuration

To work with the DTSLELA-600 use case scripts, you must have an Arm DS platform configuration containing the following:

- At least one ELA-600
- If you want to capture trace data over the ATB:
    - All the trace components linked to the ELA-600
    - All the component connections between the ELA-600 and its trace sink
- If the ELA-600 is using CTIs, add all the CTIs and CTI connections

For the Cortex-A55 + ELA-600 + CCI-500 example board, there is a platform configuration with all the necessary items already included in the DTSLELA-600 project. The following are screenshots of the platform configuration SDF file, ELA600.sdf, with the necessary components and connections outlined in red:

Note: Arm DS does not support connecting an ELA-600 to a CTI.



ELA600.sdf

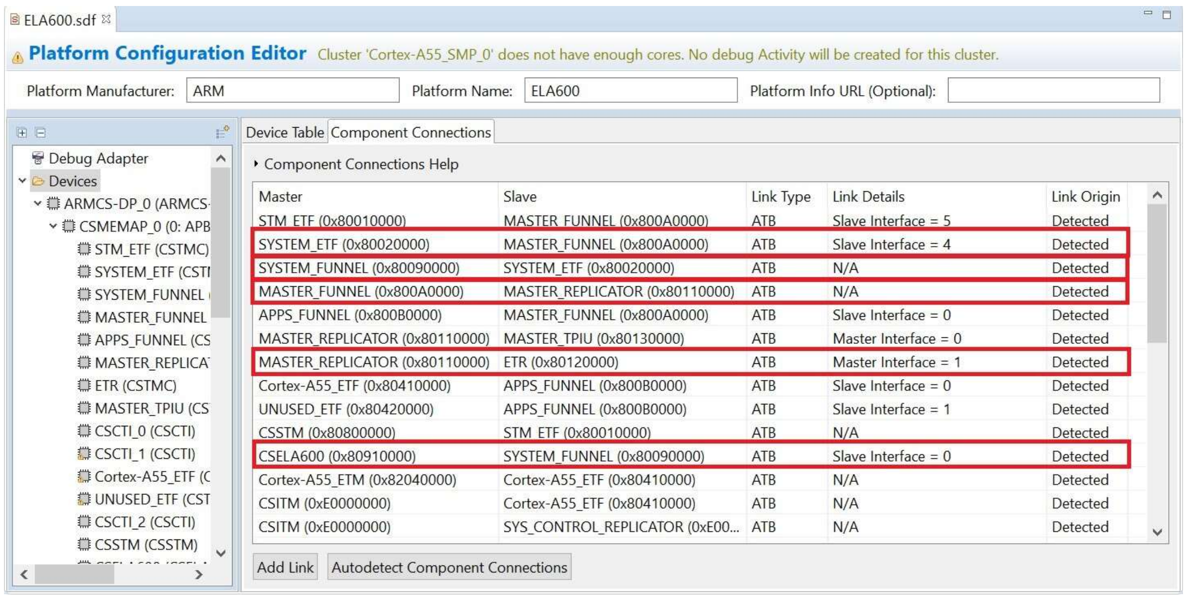
Platform Configuration Editor Cluster 'Cortex-A55_SMP_0' does not have enough cores. No debug Activity will be created for this cluster.

Platform Manufacturer: ARM Platform Name: ELA600 Platform Info URL (Optional):

- Debug Adapter
- Devices
  - ARMCS-DP_0 (ARMCS-D
    - CSMEMAP_0 (0: APB-A
      - STM_ETF (CSTMC)
      - SYSTEM_ETF (CSTMC
      - SYSTEM_FUNNEL (C
      - MASTER_FUNNEL (C
      - APPS_FUNNEL (CSTF
      - MASTER_REPLICATO
      - ETR (CSTMC)
      - MASTER_TPIU (CSTP
      - CSCTI_0 (CSCTI)
      - CSCTI_1 (CSCTI)
      - Cortex-A55_ETF (CS
      - UNUSED_ETF (CSTM
      - CSCTI_2 (CSCTI)
      - CSSTM (CSSTM)
      - CSELA600 (CSELA60

Device Table | Component Connections

RD...	Device Name	Device Type	Device Fa...	Device Class	AP Index	Base Address
4	SYSTEM_ETF	CSTMC	CoreSight	TraceSink	0	0x80020000
5	SYSTEM_FUNNEL	CSTFunnel	CoreSight	Link	0	0x80090000
6	MASTER_FUNNEL	CSTFunnel	CoreSight	Link	0	0x800A0000
7	APPS_FUNNEL	CSTFunnel	CoreSight	Link	0	0x800B0000
8	MASTER_REPLICATOR	CSATBReplicator	CoreSight	Register	0	0x80110000
9	ETR	CSTMC	CoreSight	TraceSink	0	0x80120000
10	MASTER_TPIU	CSTPIU	CoreSight	TraceSink	0	0x80130000
11	CSCTI_0	CSCTI	CoreSight	Link	0	0x80140000
12	CSCTI_1	CSCTI	CoreSight	Link	0	0x80150000
13	Cortex-A55_ETF	CSTMC	CoreSight	TraceSink	0	0x80410000
14	UNUSED_ETF	CSTMC	CoreSight	TraceSink	0	0x80420000
15	CSCTI_2	CSCTI	CoreSight	Link	0	0x80610000
16	CSSTM	CSSTM	CoreSight	TraceSource	0	0x80800000
17	CSELA600	CSELA600	CoreSight	Register	0	0x80910000
18	CSCTI_3	CSCTI	CoreSight	Link	0	0x80920000
19	Cortex-A55	Cortex-A55	Cortex	CoreExecutable	0	0x82010000
20	Cortex-A55_CTI	CSCTI	CoreSight	Link	0	0x82020000



ELA600.sdf

Platform Configuration Editor Cluster 'Cortex-A55_SMP_0' does not have enough cores. No debug Activity will be created for this cluster.

Platform Manufacturer: ARM Platform Name: ELA600 Platform Info URL (Optional):

- Debug Adapter
- Devices
  - ARMCS-DP_0 (ARMCS-
    - CSMEMAP_0 (0: APB
      - STM_ETF (CSTMC)
      - SYSTEM_ETF (CSTI
      - SYSTEM_FUNNEL
      - MASTER_FUNNEL
      - APPS_FUNNEL (CS
      - MASTER_REPLICA
      - ETR (CSTMC)
      - MASTER_TPIU (CS
      - CSCTI_0 (CSCTI)
      - CSCTI_1 (CSCTI)
      - Cortex-A55_ETF (C
      - UNUSED_ETF (CST
      - CSCTI_2 (CSCTI)
      - CSSTM (CSSTM)

Device Table | Component Connections

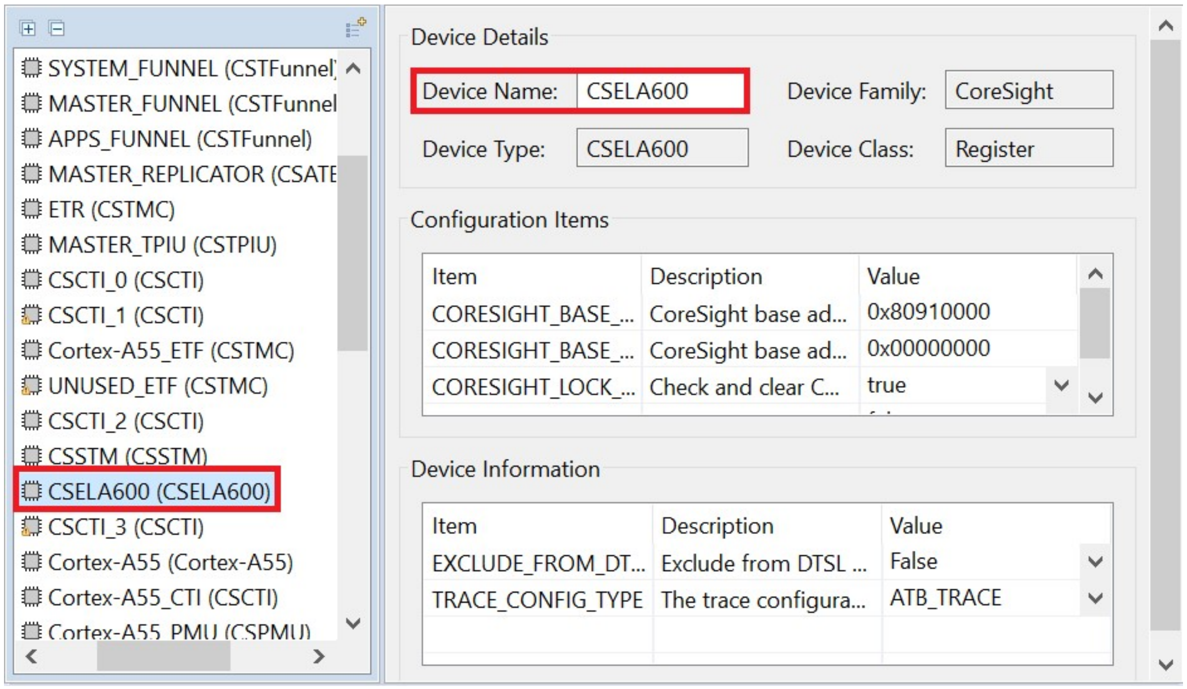
Component Connections Help

Master	Slave	Link Type	Link Details	Link Origin
STM_ETF (0x80010000)	MASTER_FUNNEL (0x800A0000)	ATB	Slave Interface = 5	Detected
SYSTEM_ETF (0x80020000)	MASTER_FUNNEL (0x800A0000)	ATB	Slave Interface = 4	Detected
SYSTEM_FUNNEL (0x80090000)	SYSTEM_ETF (0x80020000)	ATB	N/A	Detected
MASTER_FUNNEL (0x800A0000)	MASTER_REPLICATOR (0x80110000)	ATB	N/A	Detected
APPS_FUNNEL (0x800B0000)	MASTER_FUNNEL (0x800A0000)	ATB	Slave Interface = 0	Detected
MASTER_REPLICATOR (0x80110000)	MASTER_TPIU (0x80130000)	ATB	Master Interface = 0	Detected
MASTER_REPLICATOR (0x80110000)	ETR (0x80120000)	ATB	Master Interface = 1	Detected
Cortex-A55_ETF (0x80410000)	APPS_FUNNEL (0x800B0000)	ATB	Slave Interface = 0	Detected
UNUSED_ETF (0x80420000)	APPS_FUNNEL (0x800B0000)	ATB	Slave Interface = 1	Detected
CSSTM (0x80800000)	STM_ETF (0x80010000)	ATB	N/A	Detected
CSELA600 (0x80910000)	SYSTEM_FUNNEL (0x80090000)	ATB	Slave Interface = 0	Detected
Cortex-A55_ETM (0x82040000)	Cortex-A55_ETF (0x80410000)	ATB	N/A	Detected
CSITM (0xE0000000)	Cortex-A55_ETF (0x80410000)	ATB	N/A	Detected
CSITM (0xE0000000)	SYS_CONTROL_REPLICATOR (0xE00...	ATB	N/A	Detected

Add Link | Autodetect Component Connections

Note: All the Arm DS DTSL ELA-600 use case scripts assume the ELA-600 is named "CSELA600" in the platform configuration SDF file. To use your platform configuration with the Arm DS use case scripts, you either must:

- In the SDF file, change the ELA-600 component Device Name to "CSELA600". Save the change with File > Save.



Device Details

Device Name:	CSELA600	Device Family:	CoreSight
Device Type:	CSELA600	Device Class:	Register

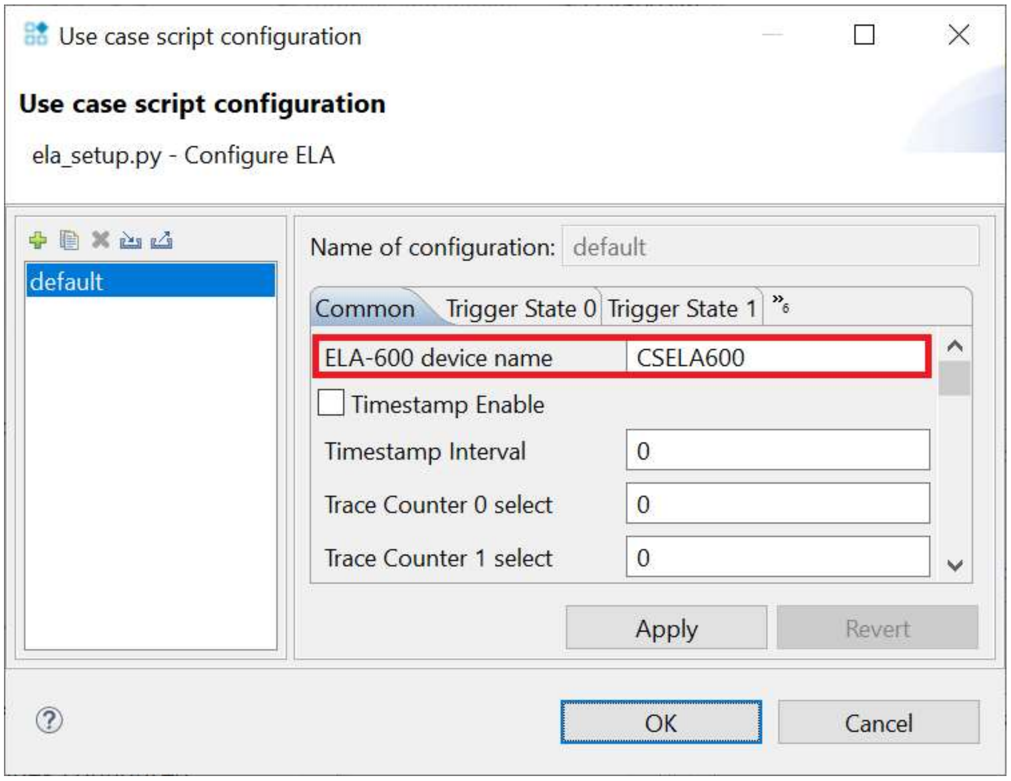
Configuration Items

Item	Description	Value
CORESIGHT_BASE_...	CoreSight base ad...	0x80910000
CORESIGHT_BASE_...	CoreSight base ad...	0x00000000
CORESIGHT_LOCK_...	Check and clear C...	true

Device Information

Item	Description	Value
EXCLUDE_FROM_DT...	Exclude from DTSL ...	False
TRACE_CONFIG_TYPE	The trace configura...	ATB_TRACE

- Manually configure the ELA-600 device name field for each use case script to the Device Name specified in the SDF file. Save the use case script changes by clicking Apply and OK. As an example of one of the scripts to change, the following shows the ELA-600 device name field in the ela_setup.py – Configure ELA use case script.


Use case script configuration

Use case script configuration

ela_setup.py - Configure ELA

Name of configuration: default

ELA-600 device name	CSELA600
☐ Timestamp Enable	
Timestamp Interval	0
Trace Counter 0 select	0
Trace Counter 1 select	0

5 Setting up the DTSL options

To work with the DTSLELA-600 use case scripts, as part of the debug connection, you must set up the Arm DS Debugger Debug and Trace Services Layer (DTSL) Configuration view. This tutorial uses a bare-metal debug hardware connection to the platform configuration included with the DTSLELA-600 project.

To learn how to create a debug configuration for your board, read the "Configuring debug connections in Arm Debugger" section of the Arm Development Studio User Guide.

Once you have a debug connection, open the DTSL Configuration view by doing the following:

1. If the Edit Configuration or Debug Configurations… view is not open, go to Run > Debug Configurations... .
2. In the debug connection Connection tab, if the Connections > Bare Metal Debug > Connection field is empty, manually enter the USB or TCP/IP information or Browse... for your debug probe.
3. In the Connection tab, next to DTSL Options, click Edit... .

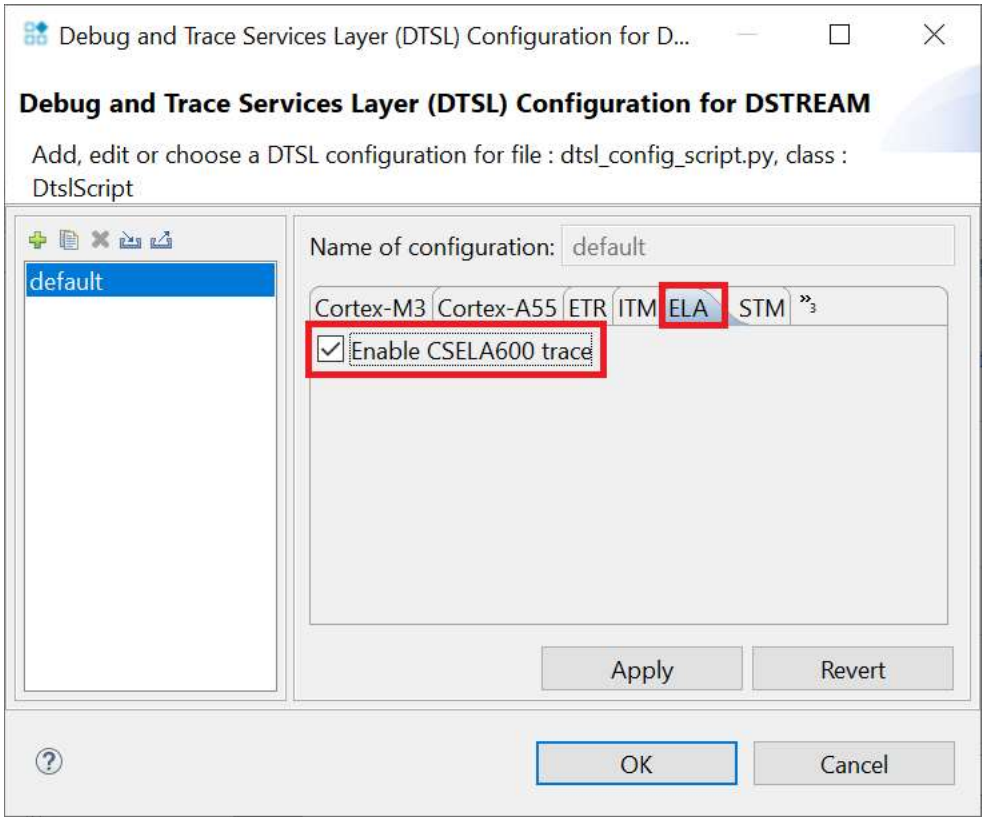
Your ELA-600 implementation can have one of three following trace data collection options:

- No trace data collection available.
- Capturing trace data to the Trace SRAM.
- Outputting trace data to the ATB for collection by a trace sink like an ETB, ETF, ETR, or TPIU.

If you are not capturing ELA-600 trace data to the ATB, only enable the ELA-600 in the DTSL Configuration view. If you do want to capture the trace data to the ATB, setup the trace components connected to the ELA-600 in the DTSL Configuration view.

For the Cortex-A55 + ELA-600 + CCI-500 system, we do the following DTSL Configuration view setup:

1. To enable the ELA-600, in the ELA tab, tick Enable CSELA600 trace.



Debug and Trace Services Layer (DTSL) Configuration for D...

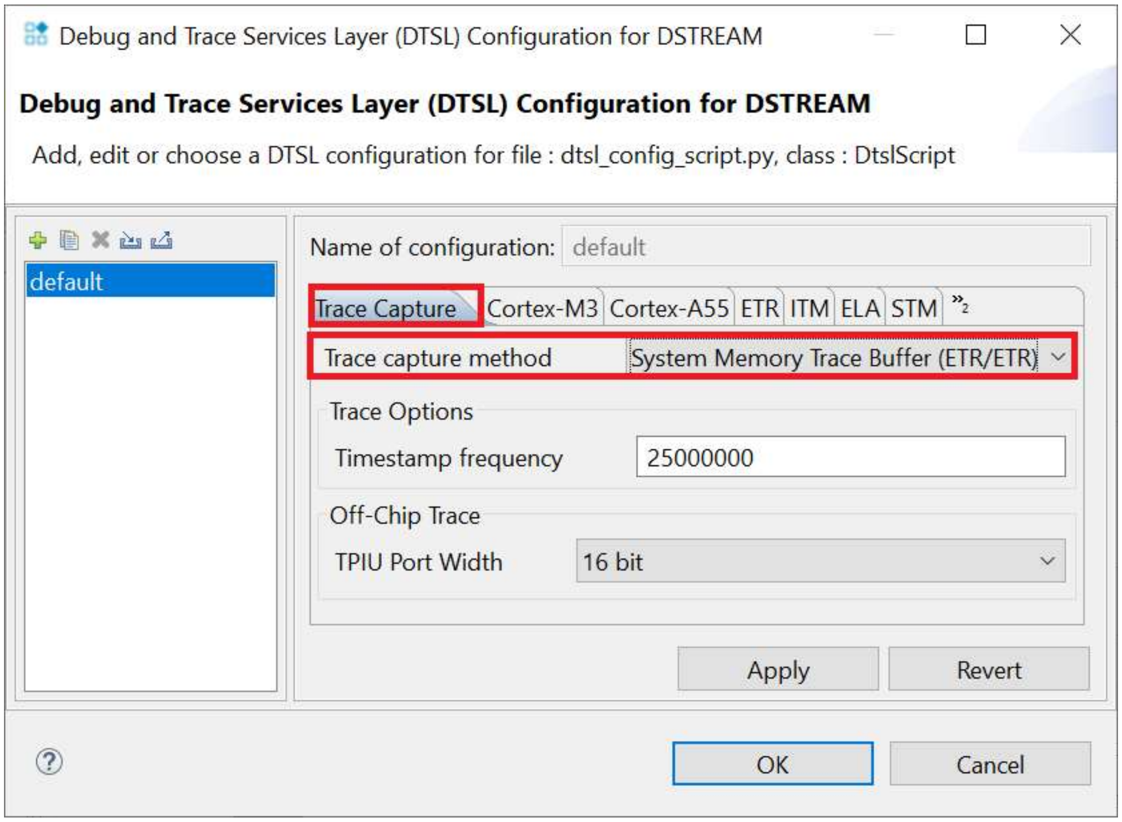
Debug and Trace Services Layer (DTSL) Configuration for DSTREAM

Add, edit or choose a DTSL configuration for file : dtsl_config_script.py, class : DtslScript

Name of configuration: default

☑ Enable CSELA600 trace

2. To enable trace data capture to the ETR, in the Trace Capture tab, set Trace capture method to System Memory Trace Buffer (ETR/ETR).


Debug and Trace Services Layer (DTSL) Configuration for DSTREAM

Debug and Trace Services Layer (DTSL) Configuration for DSTREAM

Add, edit or choose a DTSL configuration for file : dtsl_config_script.py, class : DtslScript

default

Name of configuration: default

Trace Capture | Cortex-M3 | Cortex-A55 | ETR | ITM | ELA | STM | »2

Trace capture method: System Memory Trace Buffer (ETR/ETR)

Trace Options

Timestamp frequency: 25000000

Off-Chip Trace

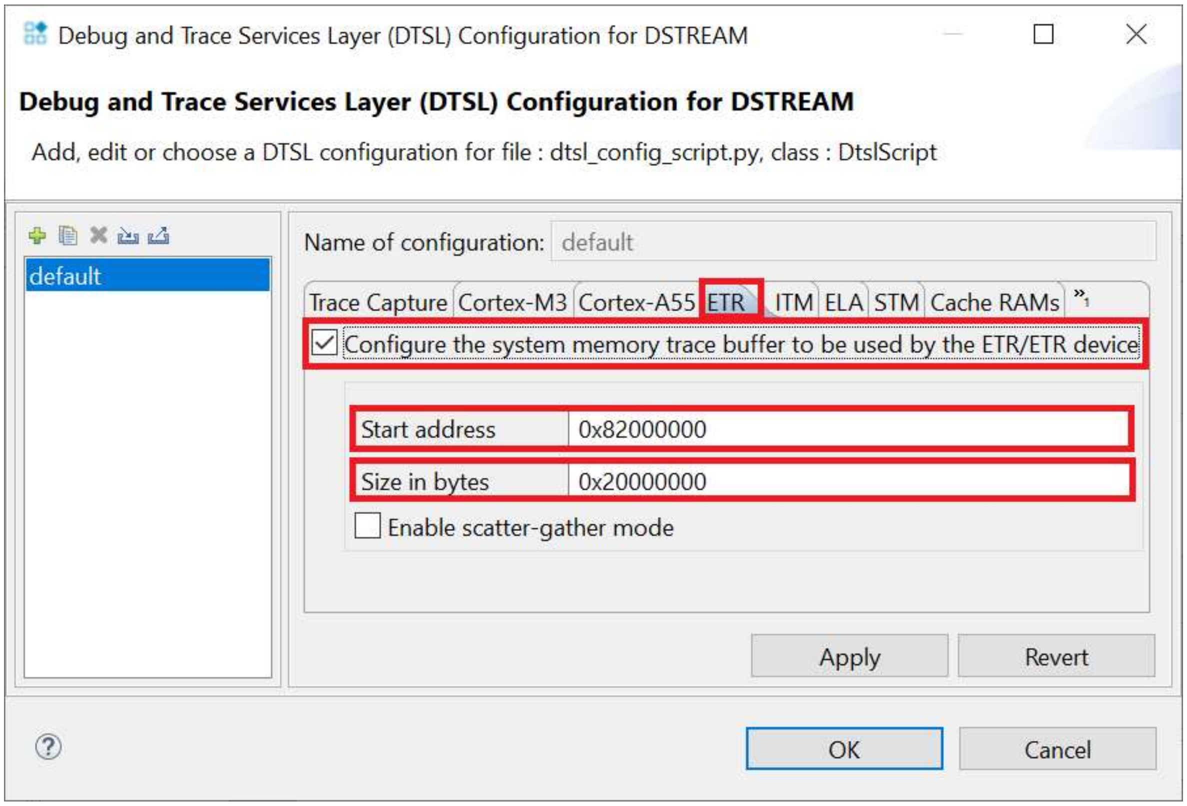
TPIU Port Width: 16 bit

3. To set up the ETR, in the ETR tab:

    a. Tick Configure the system trace memory buffer to be used by the ETR/ETR device.

    b. Set Start address to the start address of the ETR memory.

    c. Set Size in bytes to the size in bytes of the ETR memory space.



Debug and Trace Services Layer (DTSL) Configuration for DSTREAM

Add, edit or choose a DTSL configuration for file : dtsl_config_script.py, class : DtslScript

Name of configuration: default

☑ Configure the system memory trace buffer to be used by the ETR/ETR device

Start address	0x82000000
Size in bytes	0x20000000

☐ Enable scatter-gather mode

6 Configuring the ELA-600 use case scripts

To configure the ELA-600, you can 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-600. This debug recipe is achieved by programming the Common and Trigger State registers of the ELA-600. Programming the Trigger State registers sets up the comparison and counter logic to debug a scenario of interest. The Common registers set up the general configuration of the ELA-600.

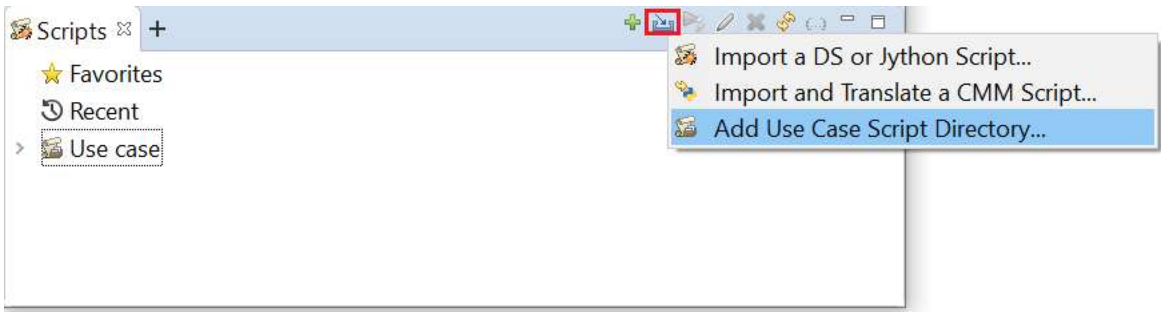
To add the DTSLELA-600 project use case scripts to the Scripts view in Arm DS, do the following:

1. Connect to a target with Arm DS.

   Learn how to connect to a target with Arm DS, read the "Configuring debug connections in Arm Debugger" section of the Arm Development Studio User Guide.

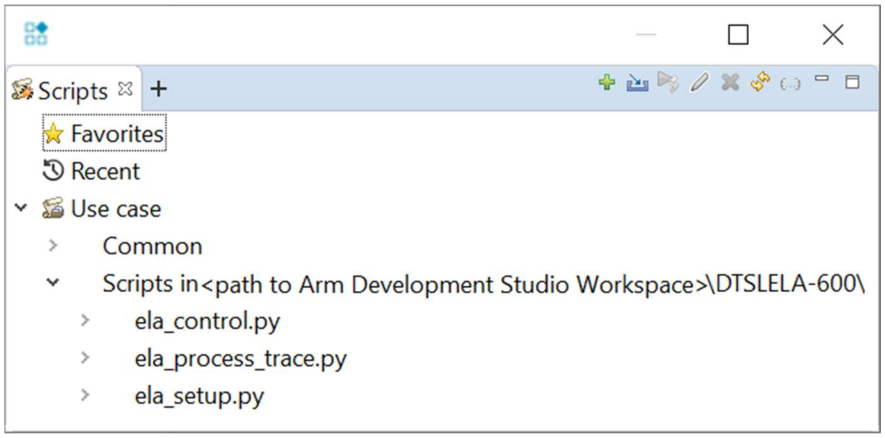
2. If the Scripts view is not open, click Window > Show View > Scripts.

3. In the Scripts view, click Import a Script or Directory > Add Use Case Script Directory....


Scripts
Favorites
Recent
Use case
Import a DS or Jython Script...
Import and Translate a CMM Script...
Add Use Case Script Directory...

4. In the Select Folder dialog, browse to the DTSLELA-600 project in your Arm DS Workspace and click Select Folder.

In the Scripts view, the DTSLELA-600 use case scripts appear under Use Case > Scripts in <path to Arm Development Studio Workspace>\DTSLELA-600\.



For this demonstration, we use the GUI ELA-600 Configuration Utility provided by the Arm DS use case script at

Scripts view > Use case > Scripts in <path to Arm Development Studio Workspace>\DTSLELA-600\ > ela_setup.py > Configure.

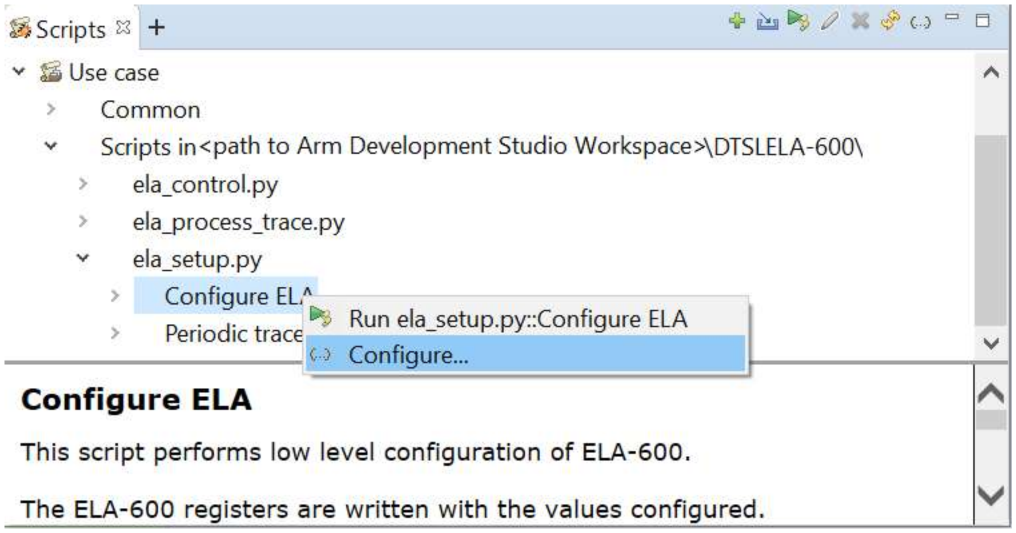
to configure the ELA-600 for our specific debug scenario.

Note: Before configuring the ELA-600, you must connect Arm DS to the target SoC.

1. Open the GUI ELA-600 configuration utility:

    a. Navigate to: Scripts view > Use case > Scripts in <path to Arm Development Studio Workspace>\DTSLELA-600\ > ela_setup.py > Configure.

    b. Right-click Configure ELA and select Configure… .



Configure ELA

This script performs low level configuration of ELA-600.

The ELA-600 registers are written with the values configured.

We now configure the Common registers:

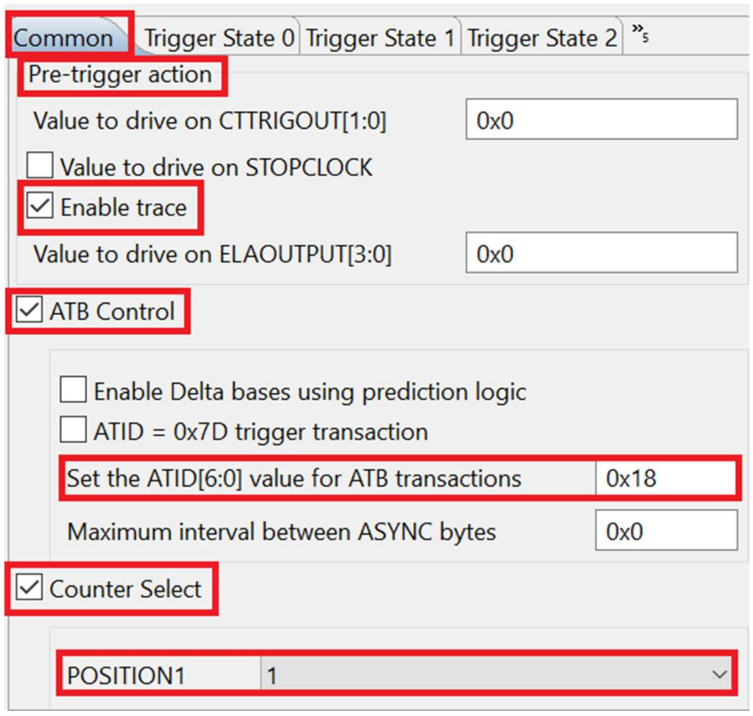
1. Open the Common tab.

2. In the **Pre-trigger action** section, select **Enable trace**.
   When the ELA-600 is enabled, this setting configures the ELA to start tracing. It sets `TRACE` of the Pre-Trigger Action register (PTACTION). When trace is active, trace capture occurs either on each ELA clock cycle, on a Trigger Signal Comparison match, or on a Trigger Counter Comparison match.

3. Tick **ATB Control** and **Set ATID[6:0] value for ATB transactions** to **0x18**.
   This setting configures the ATB trace ID for the ELA-600. It sets `ATID_VALUE` of the ATB Control register (ATBCTRL). If other trace sources are using the ATB, this value is used to identify the trace stream belonging to the ELA-600.

4. Tick **Counter Select** and set **Position 1** to **1**.
   This setting makes the trace data capture the counter value for Trigger State 1. It sets `POSITION1` of the Counter Select register (CNTSEL).

   When the preceding steps are finished, the **Common** tab looks like the following:



Pre-trigger action
Value to drive on CTTRIGOUT[1:0]: 0x0
☐ Value to drive on STOPCLOCK
☑ Enable trace
Value to drive on ELAOUTPUT[3:0]: 0x0
☑ ATB Control
☐ Enable Delta bases using prediction logic
☐ ATID = 0x7D trigger transaction
Set the ATID[6:0] value for ATB transactions: 0x18
Maximum interval between ASYNC bytes: 0x0
☑ Counter Select
POSITION1: 1

5. Click **Apply**.

We now configure our first Trigger State:

1. Open the **Trigger State 0** tab.

2. Set **Select Signal Group** to **0x1**.
   This setting selects the Signal Group we want to trigger on for the Trigger State. This configuration means that Trigger State 0 is associated with the signals in Signal Group 0. It sets Signal Select 0 register (SIGSEL0). The ELA-600 uses a 'ones hot' encoding for the Signal Group in the Signal Select registers.

In our example, Cortex-A55 + ELA-600 + CCI-500, the `VALID_P1`, `Address_P1`, and `Type_P1` signals reside in Signal Group 0. To locate the wanted signal locations for other targets, check your IP JSON file or documentation.

3. In the Trigger Control section:

    a. Set Signal Comparison (COMP) to Equal.
    This setting configures the Signal Comparison condition for the Trigger State. It sets `COMP` of the Trigger Control 0 register (TRIGCTRL0). In this case, we want to trigger when:

    - `VALID_P1` is `1`.
    - `Address_P1` is `0xB1000000`.
    - `Type_P1` is `1`.

    b. Set Comparison mode (COMPSEL) to 1.
    This setting configures the Trigger State 0 counters and selects Trigger Counter Comparison mode. It sets `COMPSEL` of the TRIGCTRL0 register.

    c. Set Counter source (COUNTSRC) to 1.
    This setting causes the Trigger State 0 counter to increment on every Trigger Signal Comparison Match. It sets `COUNTSRC` of the TRIGCTRL0 register.

4. Set Next State to 0x2.
Here we set the Next State. This is the ELA state we enter when we meet the Trigger Condition. It sets `NEXTSTATE0` of the Next State 0 register (NEXTSTATE0). In this example, when the Trigger Condition is met, we want the ELA-600 to move to Trigger State 1.

5. In the Action section:

    a. Set the CTTRIGOUT[1:0] value to 0x1.
    This setting has Trigger State 0 drive CTTRIGOUT[0] when moving to the next Trigger State. It sets `CTTRIGOUT` of the Action 0 register (ACTION0). In our case, driving CTTRIGOUT[0] corresponds to a halt signal on the core. When the Trigger State 0 Trigger Condition is met, this halt signal causes the core to halt.

    b. Tick Enable trace.
    This setting enables Trigger State 0 trace capture. It sets `TRACE` of the ACTION0 register.

6. Set Counter compare to 0x3.
This setting configures `COUNTCOMP0` of the Counter Compare 0 register (COUNTCOMP0). With this setting, after 3 Trigger Conditions are met, the ELA-600 moves to Trigger State 1. In our case, the third access to the target address is the data corruption we are monitoring for.

7. Set both the Signal Mask and Signal Compare fields to the following:

   a. Set the [31:0] value to 0x0.

   b. Set the [63:32] value to 0x0.

   c. Set the [95:64] value to 0x200.

   d. Set the [127:96] value to 0x80000162.

   We must set Trigger State 0 Signal Compare and Signal Mask values for Signal Group 0 to monitor the `VALID_P1`, `Address_P1`, and `Type_P1` signals. It sets `SIGCOMP` of the Signal Compare 0 registers (SIGCOMP0) and `SIGMASK` of the Signal Mask 0 registers (SIGMASK0).

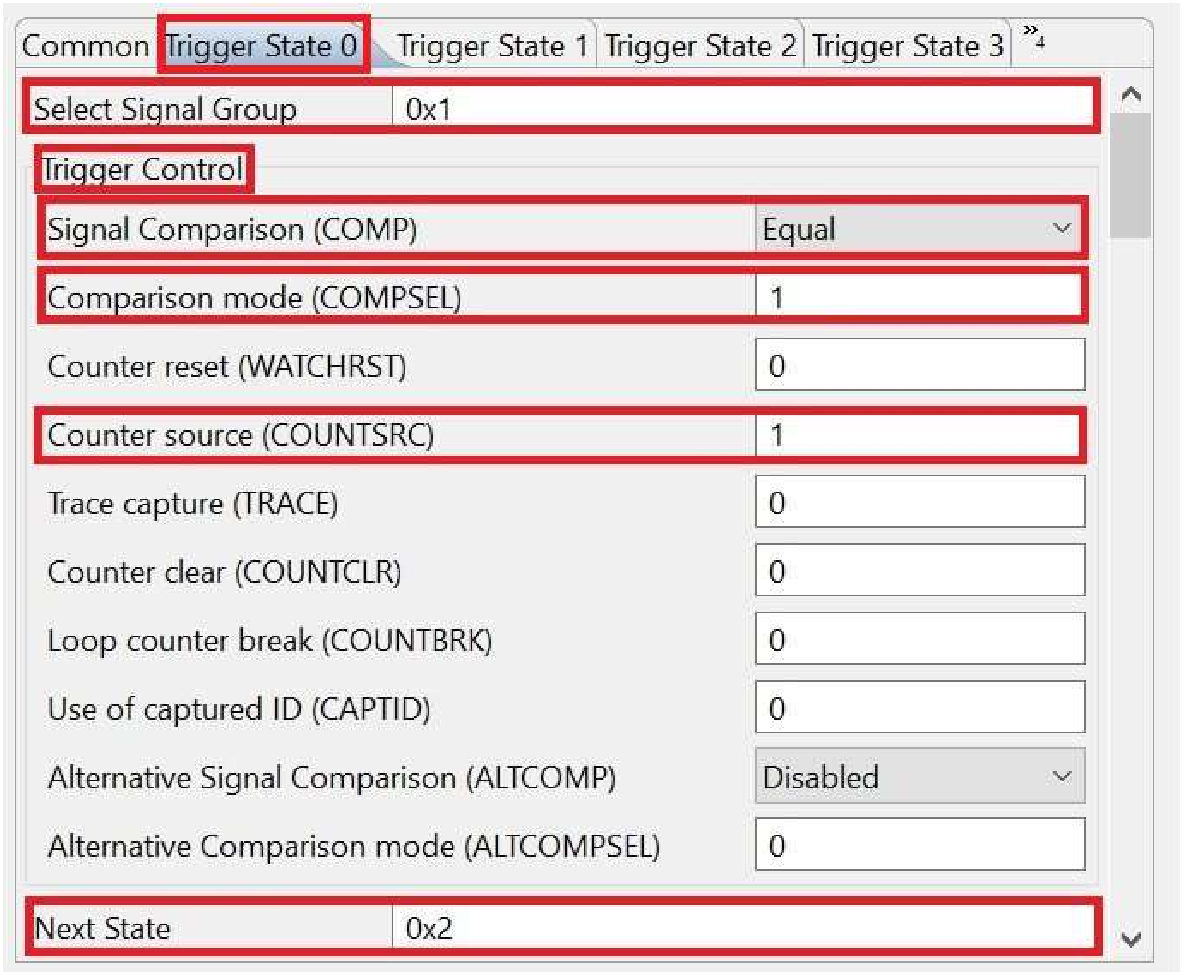
   For the Cortex-A55 + ELA-600 + CCI-500 system, the bit positions are:

   - `VALID_P1` at bit position 127
   - `Address_P1` at bit position 114:75
   - `Type_P1` at bit position 73.

   Note: You must check where these signals are positioned in the JSON file or documentation for your IP.

8. Tick Trace Write Byte Select and set TWBSEL to 0x0000FFFF.
   This setting enables trace write for each byte of Signal Group 0 we want to trace. It sets `TRACE_BYTE` of the Trace Write Byte Select 0 register (TWBSEL0).

When the preceding steps are finished, the Trigger State 0 tab looks like the following:

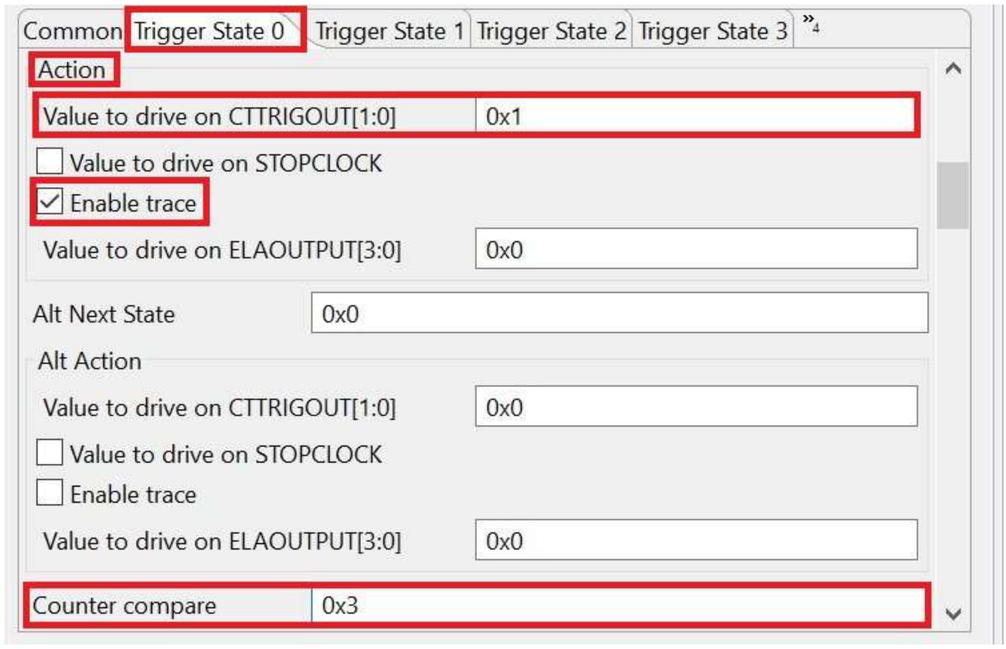


Select Signal Group: 0x1

Trigger Control

Signal Comparison (COMP)	Equal
Comparison mode (COMPSEL)	1
Counter reset (WATCHRST)	0
Counter source (COUNTSRC)	1
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

Next State: 0x2



Action

Value to drive on CTTRIGOUT[1:0]: 0x1

☐ Value to drive on STOPCLOCK

☑ Enable trace

Value to drive on ELAOUTPUT[3:0]: 0x0

Alt Next State: 0x0

Alt Action

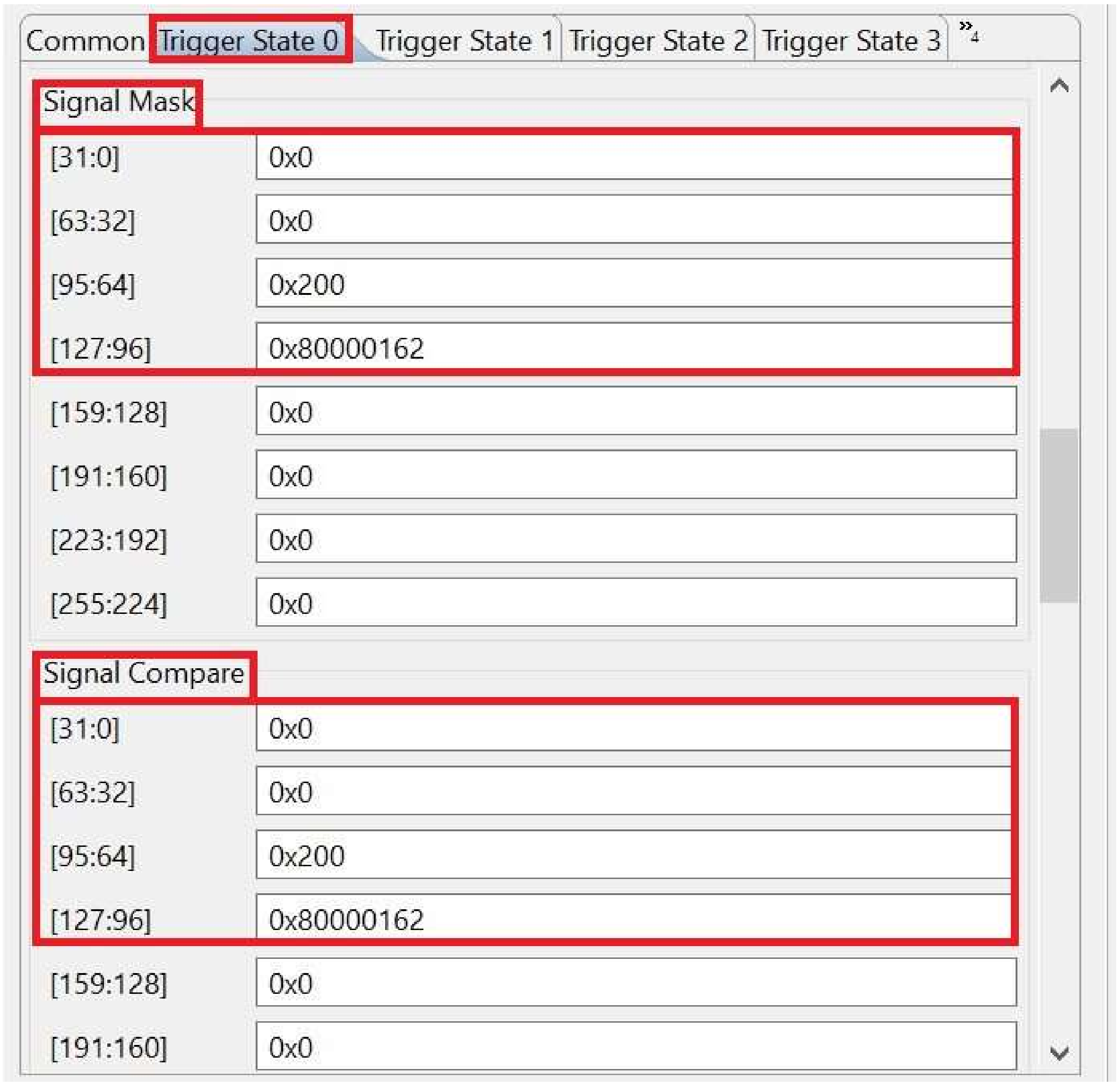
Value to drive on CTTRIGOUT[1:0]: 0x0

☐ Value to drive on STOPCLOCK

☐ Enable trace

Value to drive on ELAOUTPUT[3:0]: 0x0

Counter compare: 0x3

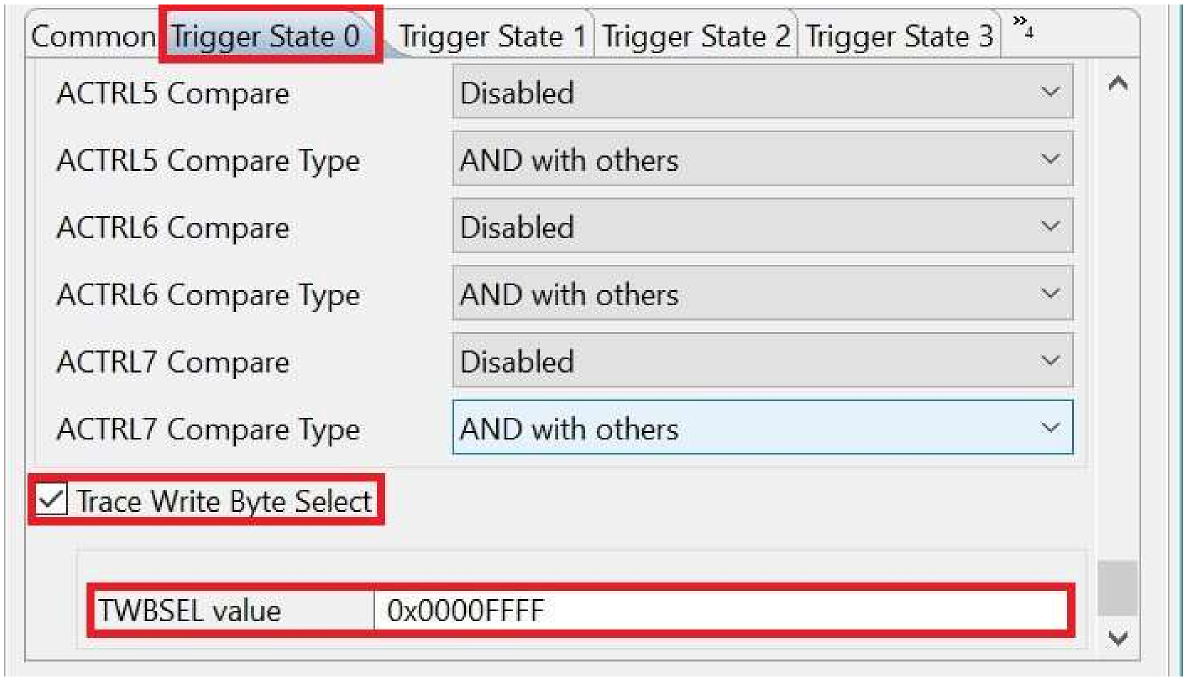


Signal Mask

[31:0]	0x0
[63:32]	0x0
[95:64]	0x200
[127:96]	0x80000162
[159:128]	0x0
[191:160]	0x0
[223:192]	0x0
[255:224]	0x0

Signal Compare

[31:0]	0x0
[63:32]	0x0
[95:64]	0x200
[127:96]	0x80000162
[159:128]	0x0
[191:160]	0x0



ACTRL5 Compare	Disabled
ACTRL5 Compare Type	AND with others
ACTRL6 Compare	Disabled
ACTRL6 Compare Type	AND with others
ACTRL7 Compare	Disabled
ACTRL7 Compare Type	AND with others

✓ Trace Write Byte Select

TWBSEL value	0x0000FFFF

9. Click Apply.

We now configure our second Trigger State:

1. Open the Trigger State 1 tab.

2. Set Select Signal Group value to 0x0.

   It sets the SIGSEL1 register. This setting causes Trigger State 1 to not trigger on any Signal Group. We do not want to trigger on any Signal Group because we are using Trigger State 1 to wait for External Trigger input.

3. In the Trigger Control section:

   a. Set Signal Comparison (COMP) to Equal.
      This setting configures the Signal Comparison condition. It sets the TRIGCTRL1 register. In this case, we want to trigger when an External Trigger Signal from the CTI comes in.

   b. Set Trace Capture (TRACE) to 3.
      This setting causes Trigger State 1 to trace the Counter Comparison. It sets `TRACE` of the TRIGCTRL1 register.

4. Set Next State to 0x0.
   We set the Next State to 0, so that Trigger State 1 is our final Trigger State. It sets the NEXTSTATE1 register.

5. In the Action section, tick Enable trace.
   This enables Trigger State 1 trace capture. It sets the ACTION1 register.

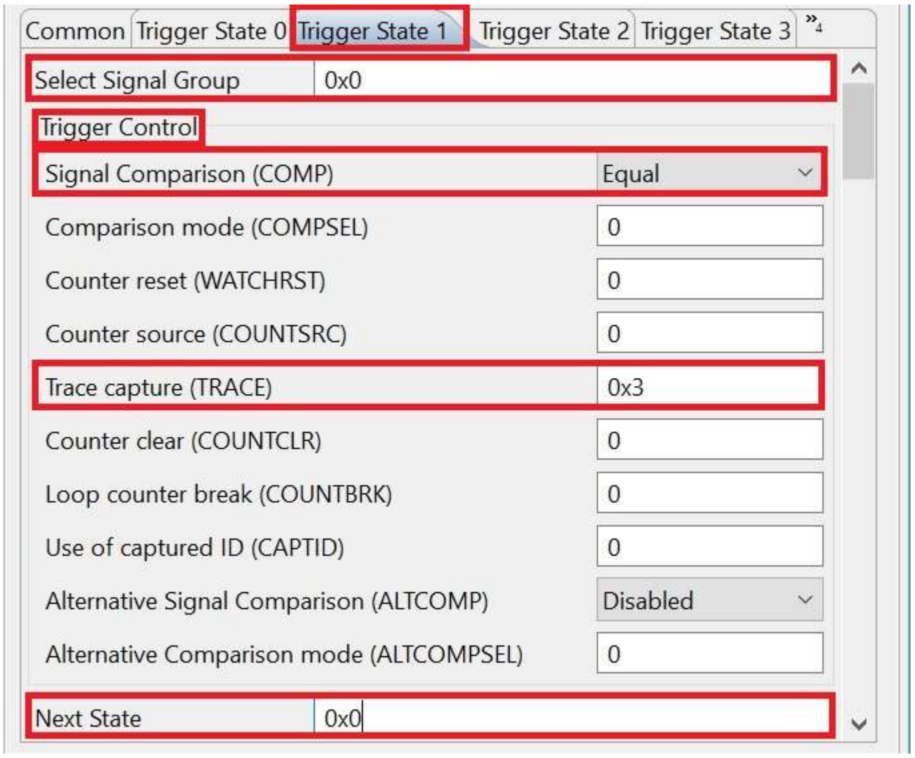
6. Set Counter compare to 0xFFFFFFFF.
   This setting configures the COUNTCOMP1 register to a nonzero value to ensure Trigger State 1 counts. We are using the maximum value allowed as the ultimate count value is unknown in this case.

7. In both the External Mask and External Compare sections, set CTTRIGIN[1:0] to 0x1.
   These settings configure a comparison with the external signal associated with the CTIIN[0] bit. It sets `CTTRIGIN` of the External Mask 1 register (EXTMASK1) and the External Compare 1 register (EXTCOMP1). In our case, CTIIN[0] is the halt response from the core.

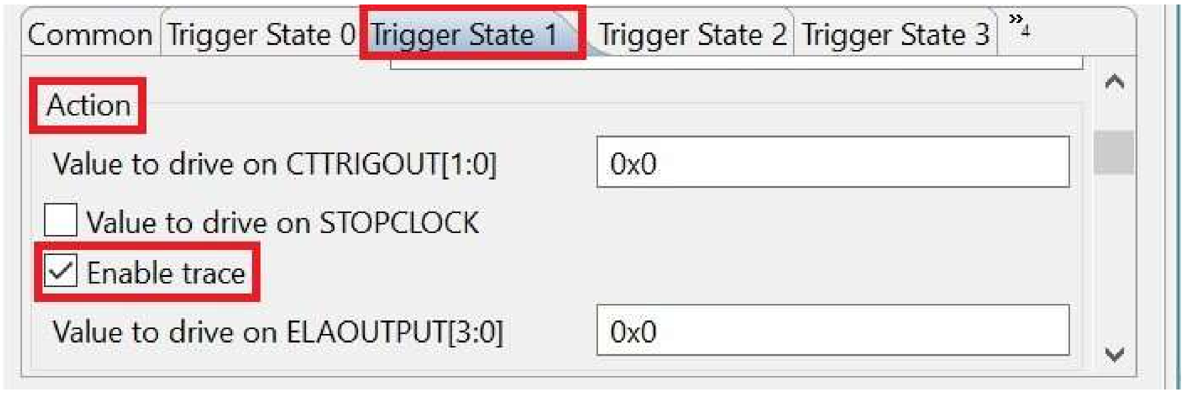
8. Set both the Signal Mask and Signal Compare fields all to 0x0.
   We must set these fields to 0, so we do not match on any of the signals from any of the Signal Groups. It sets the SIGCOMP1 and the SIGMASK1 registers.

9. Tick Trace Write Byte Select and set TWBSEL to 0x0000FFFF.
   This setting enables trace write for the data we are interested in. It sets the TWBSEL1 register.

   When the preceding steps are finished, the Trigger State 1 tab looks like the following:



Select Signal Group	0x0
Signal Comparison (COMP)	Equal
Comparison mode (COMPSEL)	0
Counter reset (WATCHRST)	0
Counter source (COUNTSRC)	0
Trace capture (TRACE)	0x3
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
Next State	0x0



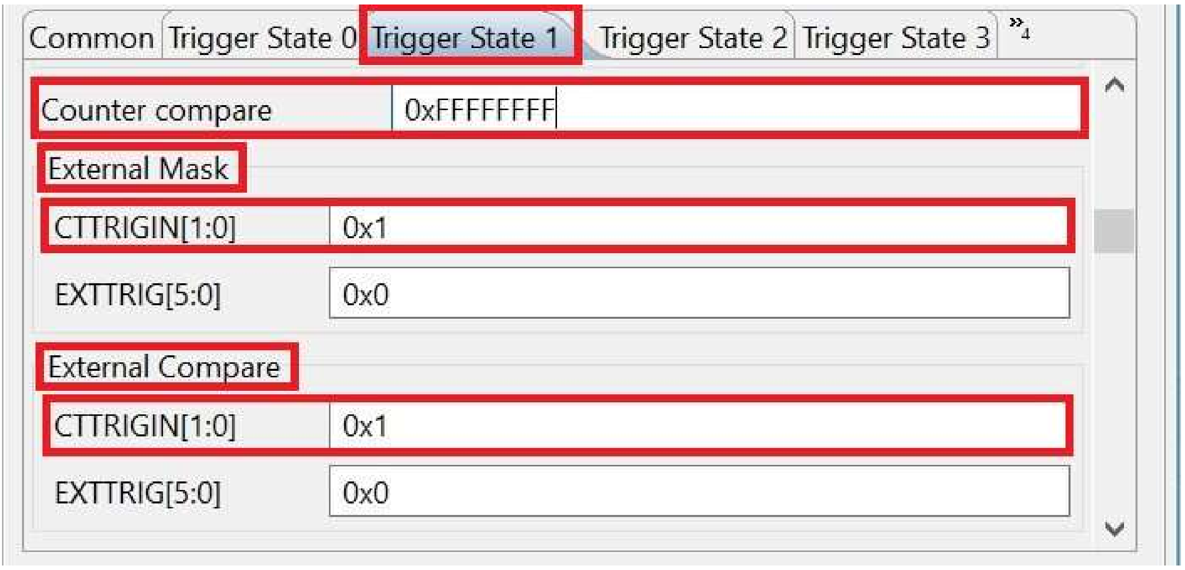
Action

Value to drive on CTTRIGOUT[1:0] 0x0

☐ Value to drive on STOPCLOCK

✓ Enable trace

Value to drive on ELAOUTPUT[3:0] 0x0



Counter compare 0xFFFFFFFF

External Mask

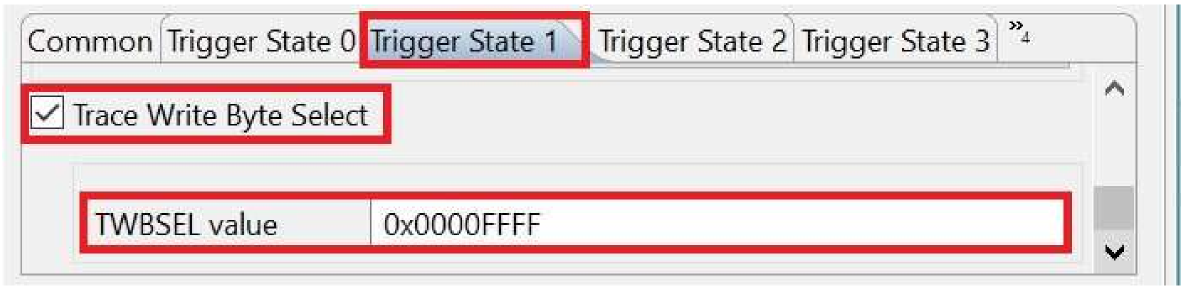
CTTRIGIN[1:0] 0x1

EXTTRIG[5:0] 0x0

External Compare

CTTRIGIN[1:0] 0x1

EXTTRIG[5:0] 0x0



✓ Trace Write Byte Select

TWBSEL value 0x0000FFFF

10. Click Apply and OK.

Alternatively, if you already have a saved ELA-600 configuration, you can import the configuration by clicking Import configuration in the ela_setup.py - Configure ELA view. An ELA-600 configuration that includes the settings mentioned in this section is available **here**.

7 Running the ELA use case scripts

1. Program the ELA configuration registers:
    1. Navigate to: Scripts view > Use case > Scripts in <path to Arm Development Studio Workspace>\DTSLELA-600\ > ela_setup.py > Configure.
    2. Right-click Configure ELA and select Run ela_setup.py::Configure ELA.
2. Set up and enable CTIs for the ELA-600 and the core.
   Setting up and enabling the CTIs allows:
    - The halt request from the Output Action of Trigger State 0 to halt the core.
    - Trigger State 1 to see the core halt response.

   We configure the CTIs by adding some code to the platform configuration dtsl_config_script.py and running a cti_setup.ds Arm DS script. Both these items are included the following Eclipse project.
3. Run the ELA:
    1. Navigate to: Scripts view > Use case > Scripts in <path to Arm Development Studio Workspace>\DTSLELA-600\ > ela_control.py > Run ELA-600.
    2. Right-click Run ELA-600 and select Run ela_setup.py::Run ELA-600.
       Run ELA-600 starts the ETR by default. If your board does use an ETR as the ELA-600 trace sink, you must:
        1. Right-click ela_control.py::Run ELA-600 and select Configure.
        2. Untick Start the connected trace sinks when the ELA-600 starts.
        3. Start the trace sink manually or add DTSL code to the Run ELA-600 use case script to start the trace sink.
4. Run the target with the test image.

Here is an Eclipse project that contains:

- Code and an image to run on the target to create the data corruption scenario.
- A init.ds Arm DS script to run as part of the target connection sequence to initialize the Cortex-A55 core.
- A cti_setup.ds Arm DS script to set up the CTI connection between the core and the ELA-600.
- A corrupt_mem.ds Arm DS script to corrupt the memory contents at address 0xB1000000.

- A dtsl_config_script.py that contains the necessary code to set up the ELA-600 CTI (CSCTI_3).

To learn how to run an Arm DS script, read the "Running a script" section of the Arm Development Studio User Guide.

Result: The target runs and the ELA monitors the input of Signal Group 0 and the External Trigger Signals for the programmed Trigger Conditions.

8 Capturing the ELA trace data

1. In this debug scenario, the core halts because the ELA-600 Trigger State 0 is programmed to send a halt request. This halt request occurs when the memory at address `0xB1000000` is corrupted. We corrupt the memory by performing a Debug Access Port (DAP) Advanced eXtensible Interface Access Port (AXI-AP) write to address `0xB1000000` and `0xB1000004` of `0xFFFFFFFF`.

   To perform these writes, run the corrupt_mem.ds Arm DS script mentioned in the preceding section. When the corrupt_mem.ds script is executed, the core is running. When the script has completed execution, the core halts.

Stop the ELA:

2. Navigate to: Scripts view > Use case > Scripts in <path to Arm Development Studio Workspace>\DTSLELA-600\ > ela_control.py > Stop ELA-600.

3. Right-click on Stop ELA-600 and select Run ela_control.py::Stop ELA-600.

   Stop ELA-600 stops the ETR by default. If your board does use an ETR as the ELA-600 trace sink, you must:

   1. Right-click ela_control.py::Stop ELA-600 and select Configure.

   2. Untick Stop the connected trace sinks when the ELA-600 stops.

   3. Stop the trace sink manually or add DTSL code to the Stop ELA-600 use case script to stop the trace sink.

Dump and decode the ELA trace:

4. Navigate to: Scripts view > Use case > Scripts in <path to Arm Development Studio Workspace>\DTSLELA-600\ > ela_process_trace.py > Decompress and decode ELA trace.

5. Right-click on Decompress and decode ELA trace and select Configure.

   1. Under the General tab, select Decompress and decode trace.

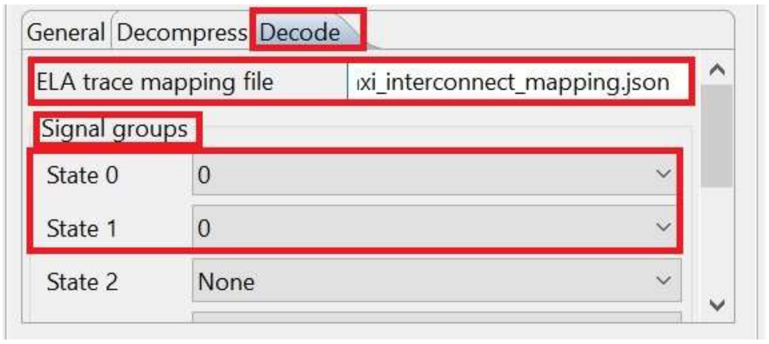
   2. Select either Output to screen or Output to file... . If there is large amount of trace data, select Output to file... and provide a file path and name.

   3. Under the Decode tab, set ELA trace mapping file to axi_interconnect_mapping.json in the DTSLELA-600 project.

      Note: If you are using your own JSON file, in ELA trace mapping file, enter the JSON file path and name.

   4. Under the Signal groups section, set State 0 and State 1 to 0.

When the preceding steps are finished, the Decode tab looks like the following:



ELA trace mapping file: ıxi_interconnect_mapping.json

Signal groups

State 0: 0

State 1: 0

State 2: None

5. Click Apply and OK.

6. Right-click on Decompress and decode ELA trace and select Run ela_process_trace.py::Decompress and decode ELA trace.

Result: If you select Output to screen in the General tab, the decompressed and decoded trace data appears in the Arm DS Commands view.

If you select Output to file... in the General tab, the decompressed and decoded trace appears in text format in the path and file you specify.

9 Analyzing the ELA trace data

As a result of the preceding steps, the ELA traces 3 transactions to address `0xB1000000` and some counter values. The trace data is output to the ETR. The 3 transactions to address `0xB1000000` are:

1. An Exclusive Load
2. A Cache Clean and Invalidate by Virtual Address to the Point of Coherency (CIVAC)
3. A store caused by the memory corruption to `0xB1000000`

The `CNTSEL[0]` counter value is the time between Trigger State 0 issuing a halt request and when the core responds with an acknowledgement signal.

The following snippet of our trace capture shows the decompressed and decoded trace data for the preceding activities.

```
Trace type: Data, Trace Stream: 0, Overrun: 0,
Data:0x80300162000003481C00400028082D07
  P1_VALID       :   1'h1
  P1_AXID        :  12'h6
  P1_addr        :  42'hB1000000
  P1non-secure   :   1'h0 => secure
  Type_P1        :   4'hD => Exclusive Read
  P0_VALID       :   1'h0
  P0_AXID        :  12'h40E
  P0_addr        :  42'h80005010
  P0non-secure   :   1'h0 => secure
  Type_P0        :   4'h2 => Read Shared, Read Clean, Read No Snoop Dirty
  TTID_P1        :   6'h34
  TTID_P0        :   6'h7
Trace type: Data, Trace Stream: 0, Overrun: 0, Data:
0xA030016200002C81C00400000602675
  P1_VALID       :   1'h1
  P1_AXID        :  12'h406
  P1_addr        :  42'hB1000000
  P1non-secure   :   1'h0 => secure
  Type_P1        :   4'hB => Write Back, Writes Clean
  P0_VALID       :   1'h0
  P0_AXID        :  12'h40E
  P0_addr        :  42'h800000C0
  P0non-secure   :   1'h0 => secure
  Type_P0        :   4'h2 => Read Shared, Read Clean, Read No Snoop Dirty
  TTID_P1         :   6'h19
  TTID_P0         :   6’h35
Trace type: Data, Trace Stream: 0, Overrun: 0, Data:
0x80400162000006506C005881F7C02C74
  P1_VALID        :   1'h1
  P1_AXID         :  12'h8
  P1_addr         :  42'hB1000000
  P1non-secure    :   1'h1 => non-secure
  Type_P1         :   4'h9 => Write No Snoop
  P0_VALID        :   1'h0
  P0_AXID         :  12'h836
  P0_addr         :  42'hB103EF80
  P0non-secure    :   1'h0 => secure
  Type_P0         :   4'h2 => Read Shared, Read Clean, Read No Snoop Dirty
  TTID_P1         :   6'h31
  TTID_P0         :   6'h34
Trace type: Counter, Trace Stream: 1, Overrun: 0, Data:
0x0000000300000003000000030000000B
CNTSEL[0] : 32'hb
CNTSEL[1] : 32'h3
CNTSEL[2] : 32'h3
CNTSEL[3] : 32’h3
```

10 Resources

Application Note - Arm CoreSight ELA-600 Version 1.0

The Arm CoreSight ELA-600 Embedded Logic Analyzer Product Page

ELA-600 Technical Reference Manual (TRM)

"Embedded Logic Analyzer (ELA)" section of the Arm Development Studio User Guide

Support for ELA-600 (a DS-5 Development Studio Video)