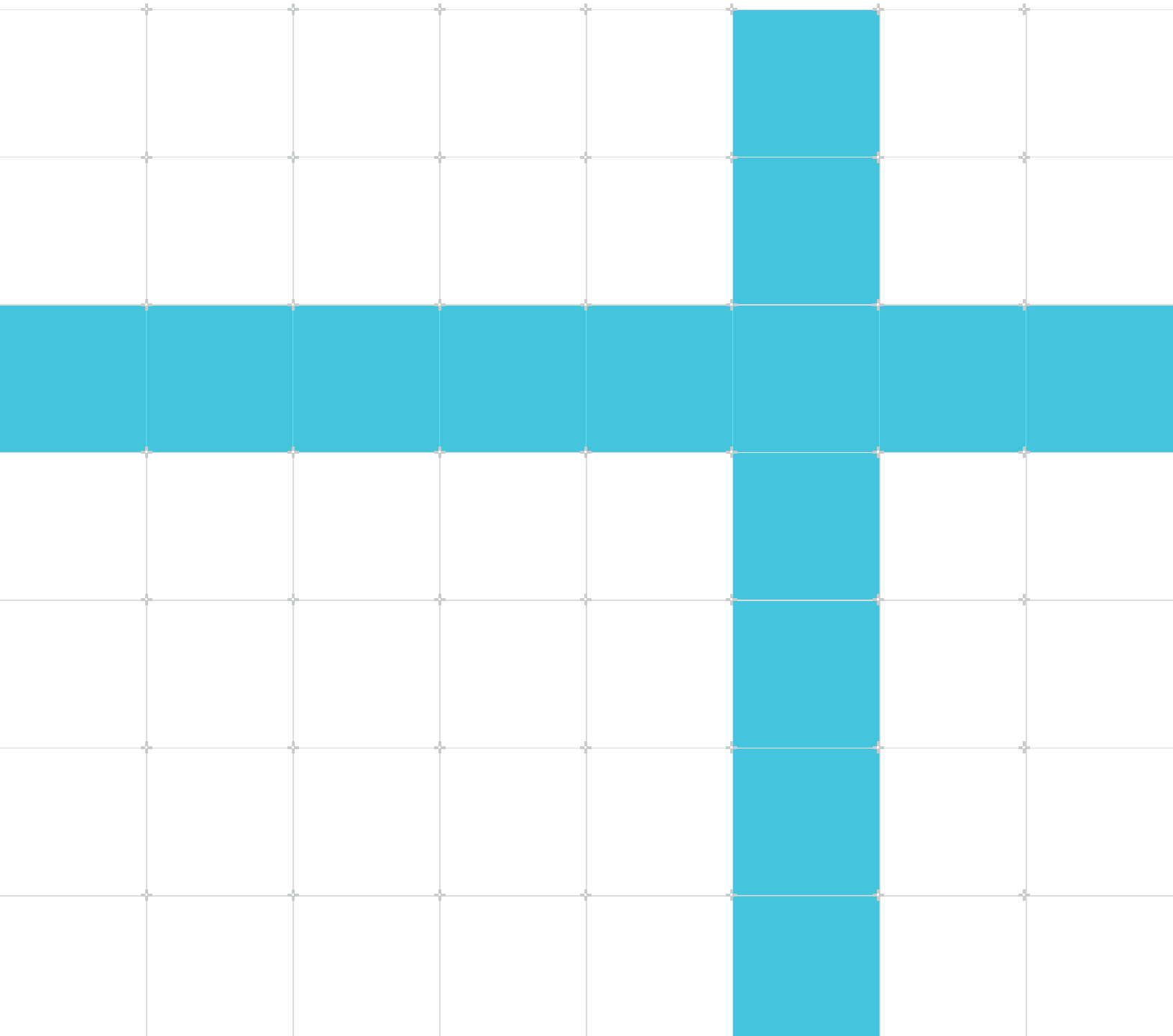
Using the CoreSight ELA-600 Embedded Logic Analyzer with Arm DS-5

Version 1.0

Issue 01
102602_0100_01_en



Using the CoreSight ELA-600 Embedded Logic Analyzer with Arm DS-5

Release information

Document history

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

Proprietary Notice

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.

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 to the CoreSight ELA-600 Embedded Logic analyzer

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 load, stores, speculative fetches, cache activity, and transaction life cycle.

CoreSight ELA-600 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.

CoreSight ELA-600 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-600 provides a superset of the functionality of the ELA-500. The largest additions being the ability to aquire trace data over the Advanced Trace Bus (ATB) and having the potential of having 8 Trigger States instead of just 5.

More information about the ELA-600 can be found on the Arm CoreSight ELA-600 Embedded Logic Analyzer Product page and in the ELA-600 Technical Reference Manual (TRM) in the Resources section of this tutorial.

The Problem

Data in memory can be corrupted by on-chip operations (like unintended code execution or stack/heap leakage) or externally (like through malicious code). So, it is usefully to have a way to monitor exactly what memory transactions are occurring for certain regions of memory and act accordingly.

In this tutorial's scenario:

- Memory accesses to a specific address (`0xB1000000`) will be tracked three times.
- The ELA-600 will then monitor and wait for any memory corruption to occur.
- When memory corruption does occur, the ELA-600 will send a halt to the core via the Cross Trigger Interface.
- The ELA-600 will count how many cycles occur between sending the halt request and the core sending an acknowledgment of the halt.

This scenario is modeled on the Data Corruption Scenario found in the Application Note - Arm CoreSight ELA-600 Version 1.0. A link to the Application Note - Arm CoreSight ELA-600 Version 1.0 can be found in the Resources section of this tutorial.

The Solution

The CoreSight ELA-600 can be used in this scenario to trace the external bus transactions made by the processor. This tutorial intends to show the use case scripting capabilities of DS-5

and demonstrate the example CoreSight ELA-600 use case scripts shipped with Arm DS-5 Development Studio.

About the CoreSight ELA-600

The ELA-600 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. An example of a JSON file can be found in the DS-5 ELA-600 deliverables (`<DS-5 installation directory>\examples\DTSL_examples.zip \DTSLELA-600\axi_interconnect_mapping.json`).

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, we will be using an example Cortex-A55 + ELA-600 + CCI-500 system. The system exposes a number of pre-defined debug observation ports (Signal Groups), and provides the corresponding JSON signal mapping file.

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

- 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 access issued by the core and by the memory corruption. Post analysis of these read transactions will allow tracking of three of the memory space transactions (the last being the memory corruption) and the counter value between the halt request and the halt request acknowledgement.

2. Before you begin

Make sure you have installed DS-5 v5.29 or later.

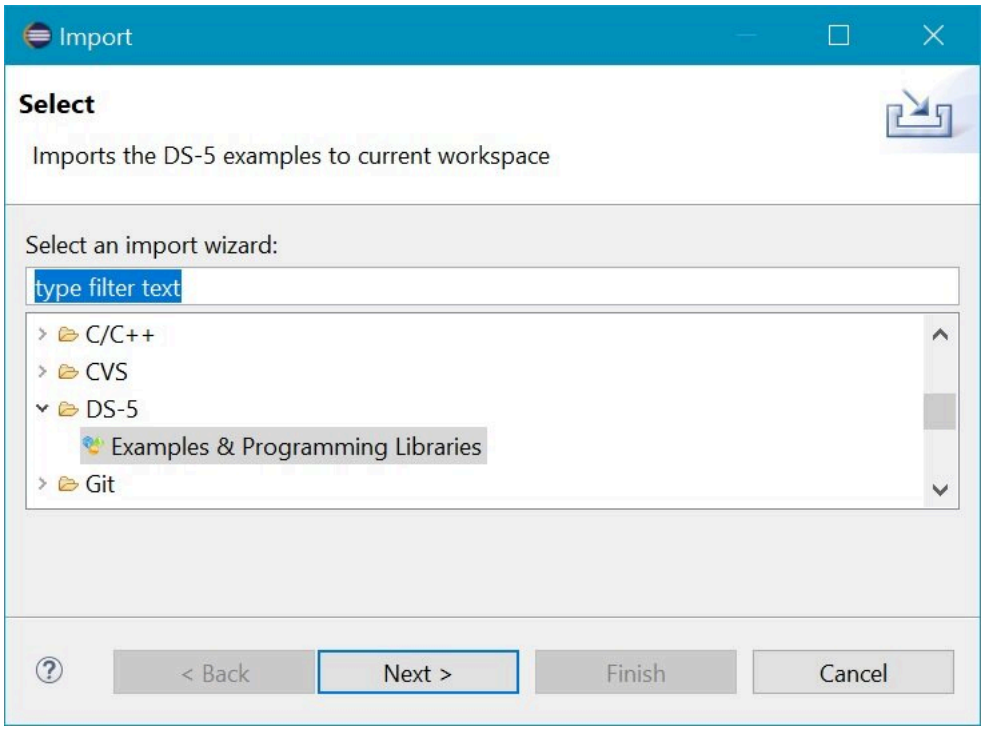
The scripts that enable DS-5 to work with the ELA-600 are available in DS-5 v5.29.

3. Importing the DTSL ELA-600 Use case scripts

With the following steps, you can import the DTSL ELA-600 Use case scripts:

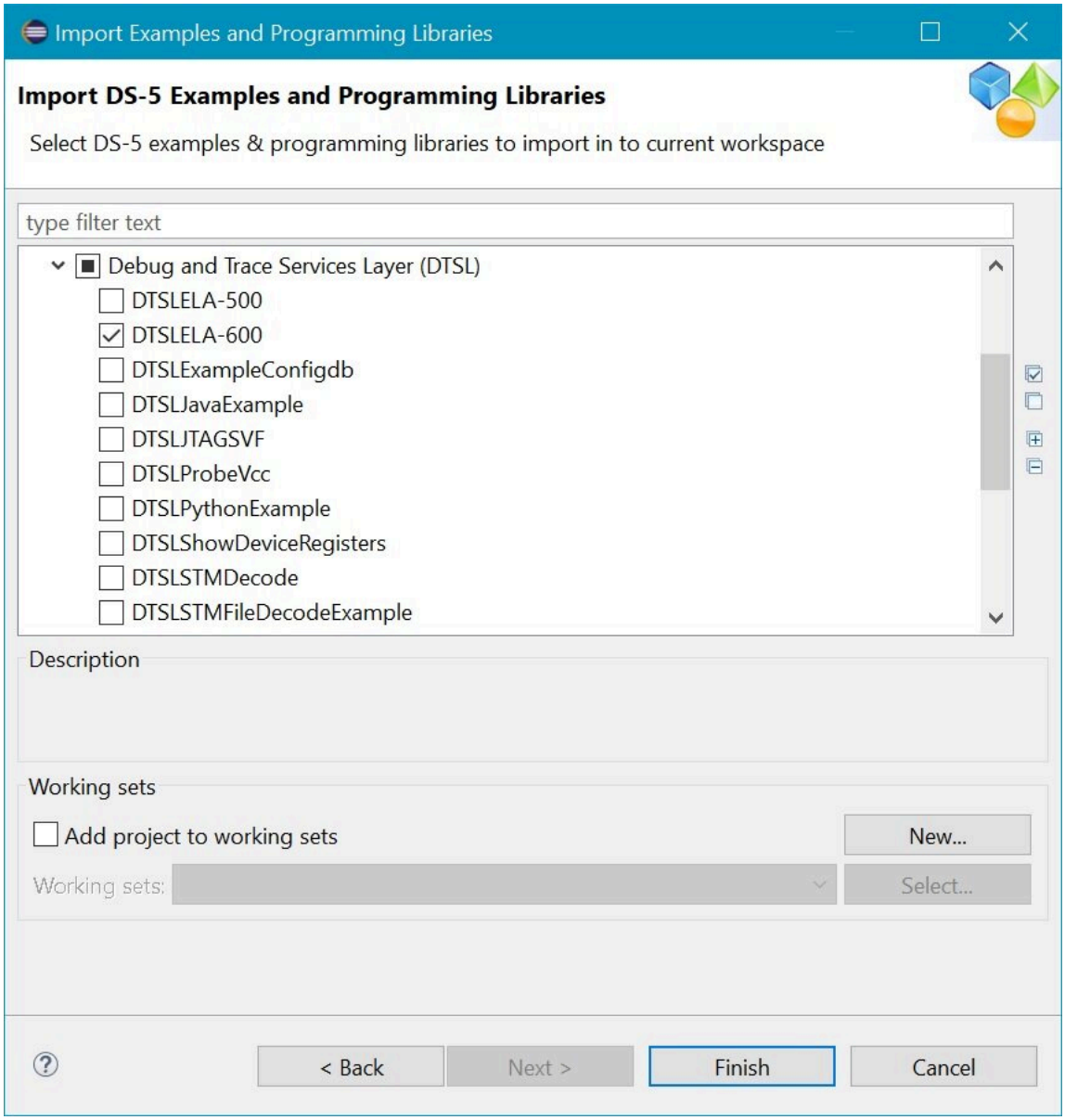
1. Launch **Eclipse for DS-5** from the **Start Menu**.
2. Select a Workspace for your DS-5 projects. The default workspace is fine.
3. Close the Welcome screen, if it appears.
4. Select **File** > **Import...** to open the **Import Selection** dialog.
5. Expand the **DS-5** group and select **Examples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

Figure 3-1: Import DS-5 Examples and Prgramming Libraries



6. Expand the **Examples** group, then expand the **Debug and Trace Services Layer (DTSL)** group.
7. Select **DTSLELA-600**.

Figure 3-2: Selecting DTSLELA-600



8. Click **Finish**.

   Result: The Project Explorer view populates with the project.

4. Getting an ELA platform configuration

To work with the DTSLELA-600 use case scripts, you will need a DS-5 platform configuration which contains:

- An 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 CTI(s), all the CTIs and CTI connections need to be present

For the Cortex-A55 + ELA-600 + CCI-500 board, there is a platform configuration with all the necessary items already included. Below is a screenshot of the platform configuration's .sdf file with the necessary components and connections outlined in red:

Figure 4-1: Platform configuration .sdf components

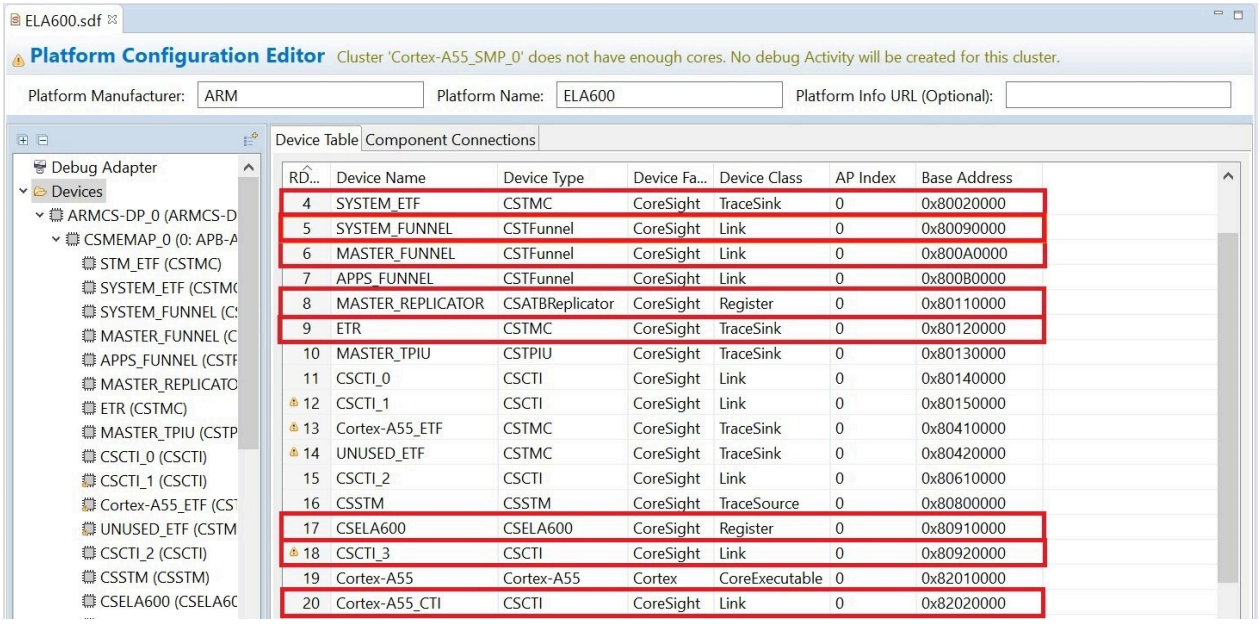
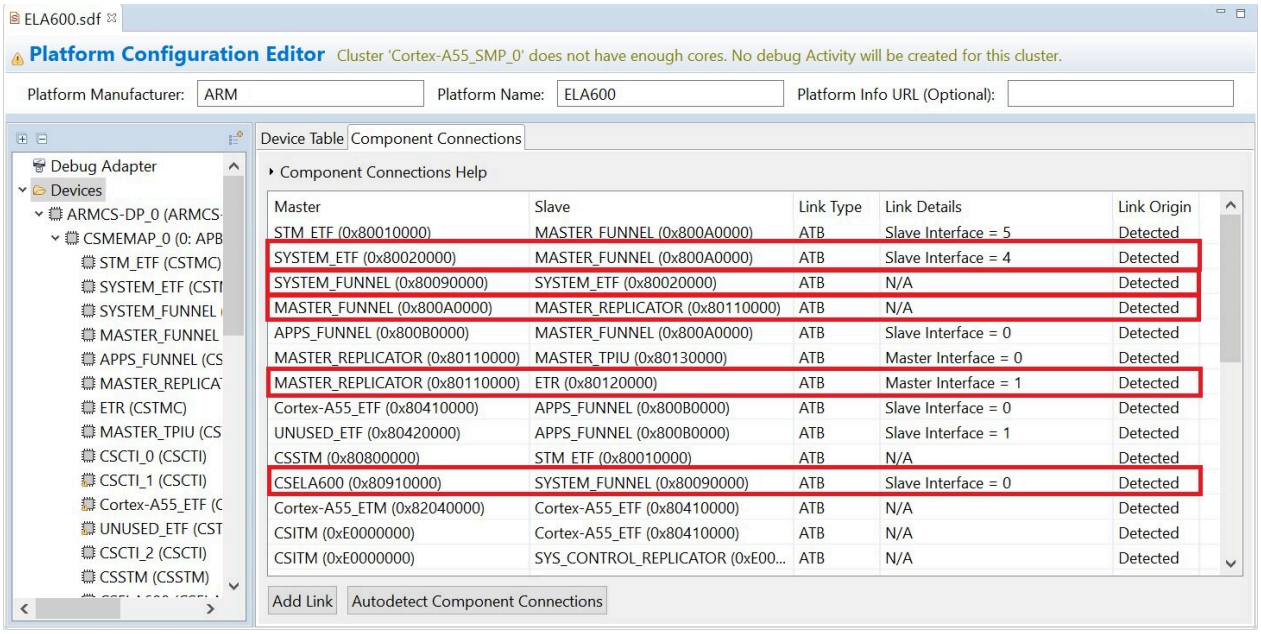


Figure 4-2: Platform configuration .sdf connections



Note

DS-5 5.29 does not support connecting an ELA-600 to a CTI. This functionality will be added in a future release.

All the DS-5 ELA-600 use case scripts assume the ELA-600 is called “CSELA600” in the platform configuration .sdf. So to use the platform configuration with the DS-5 use case scripts, you either need to change the ELA-600 name in the .sdf file to “CSELA600” if it is not already or manually configure the “ELA-600 device name” field for each use case script.

5. Setting up the DTSL options

The DTSLELA-600 use case scripts also need certain items in the DS-5 Debugger Debug and Trace Services Layer (DTSL) Configuration view to setup. You can get to the DTSL Configuration view by:

1. Going to **Run** > **Debug Configurations...**
2. Creating a new or opening an existing launch configuration under DS-5 Debugger
3. In the Connection tab, clicking the **Edit...** button next to DTSL Options

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

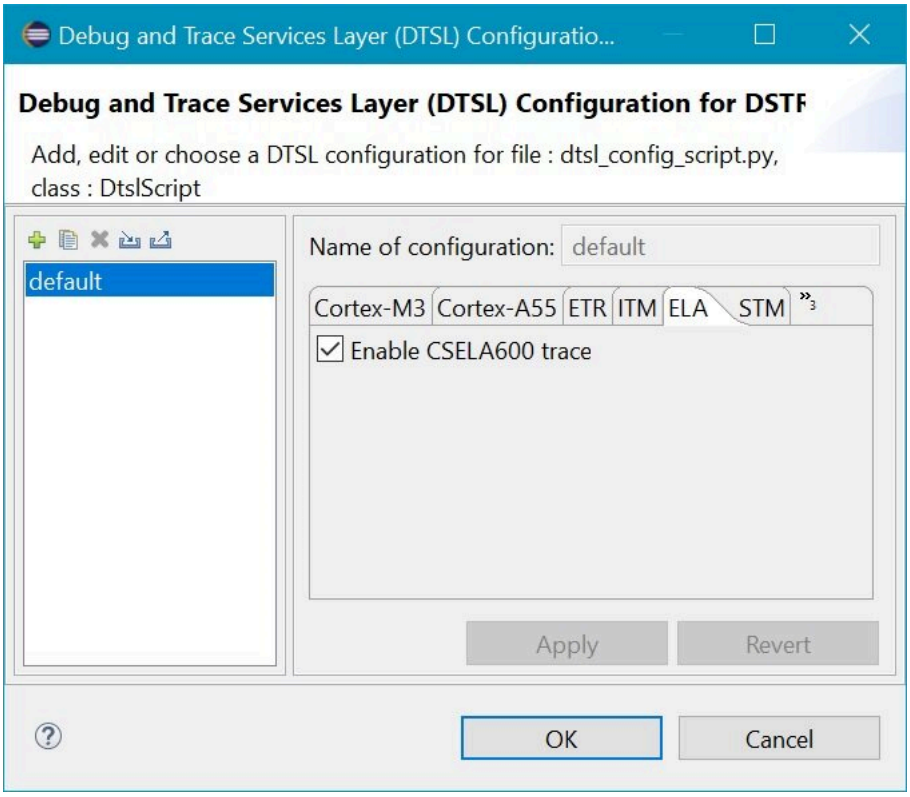
- No trace data collection available
- Capturing trace data to the Trace SRAM
- Outputting trace data to the Advanced Trace Bus (ATB) for collection by a trace sink (such as an ETB, ETF, ETR, or TPIU)

If you are not capturing ELA-600 trace data to the ATB, then you would just enable the ELA-600 in the DTSL Configurations view. If you do want to capture the trace data to the ATB, you will also need to setup the trace components connected to the ELA-600.

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

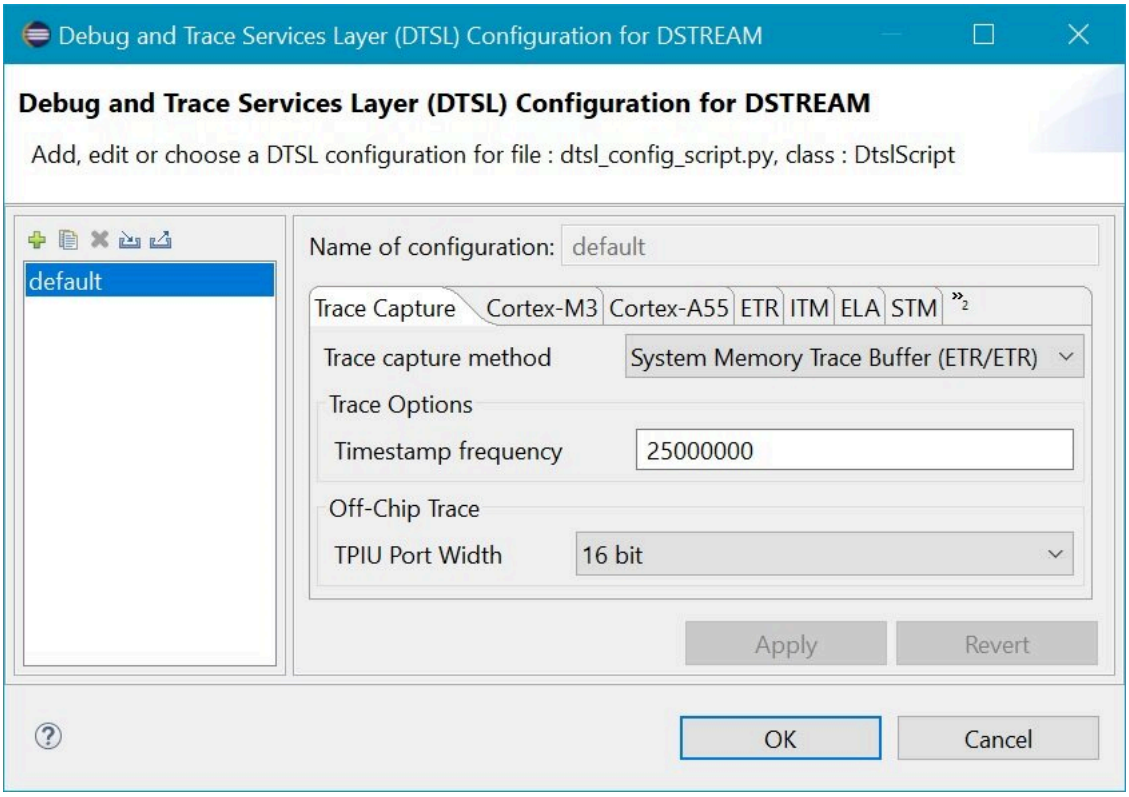
This option enables the ELA-600.

Figure 5-1: Enable the ELA-600



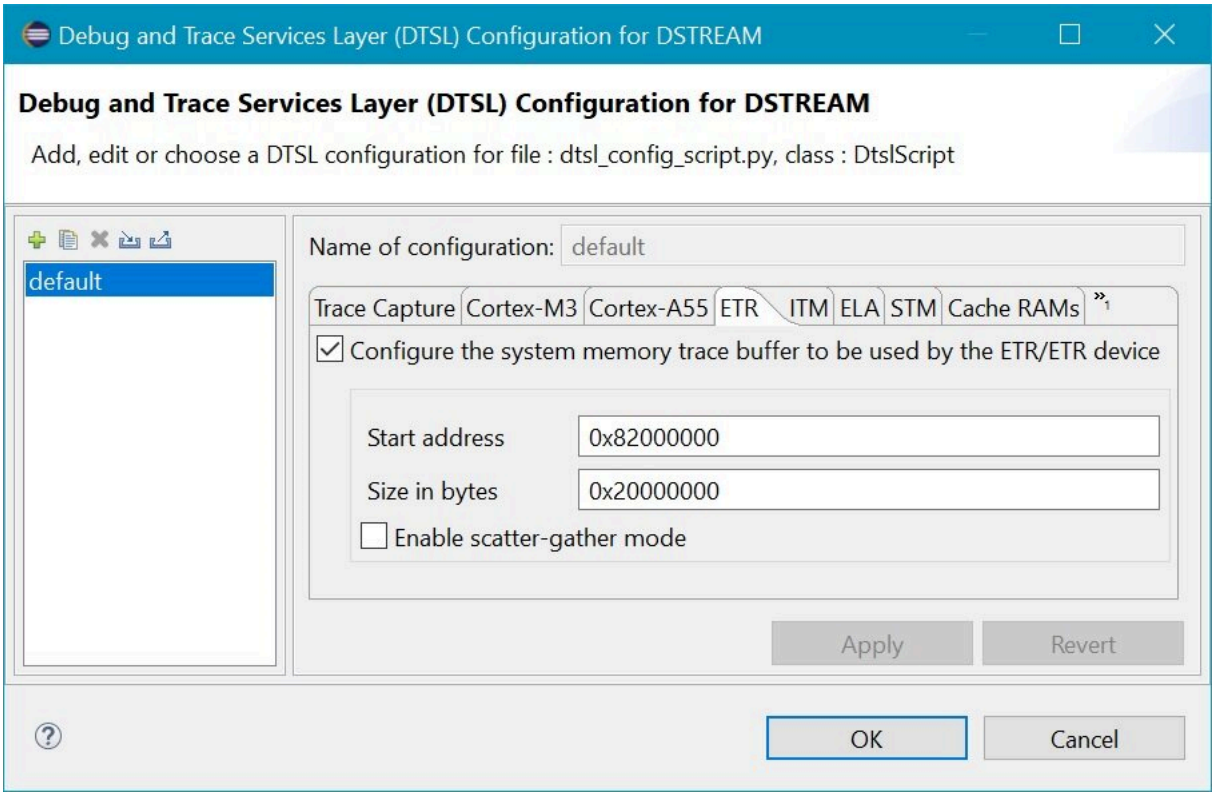
This option enables trace data capture to the ETR.

Figure 5-2: Enable trace data capture



This option sets up the ETR.

Figure 5-3: Set up the ETR



6. Configuring the ELA-600 DTSL 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/or counter logic needed to debug the scenario of interest. The Common registers are for settting up the general configuration of the ELA-600.

For this demonstration, we will use the GUI ELA-600 Configuration Utility provided by the DS-5 use case script at

Scripts window > **Use case** > **Scripts in DTSLELA-600** > **ela_setup.py** > **Configure**

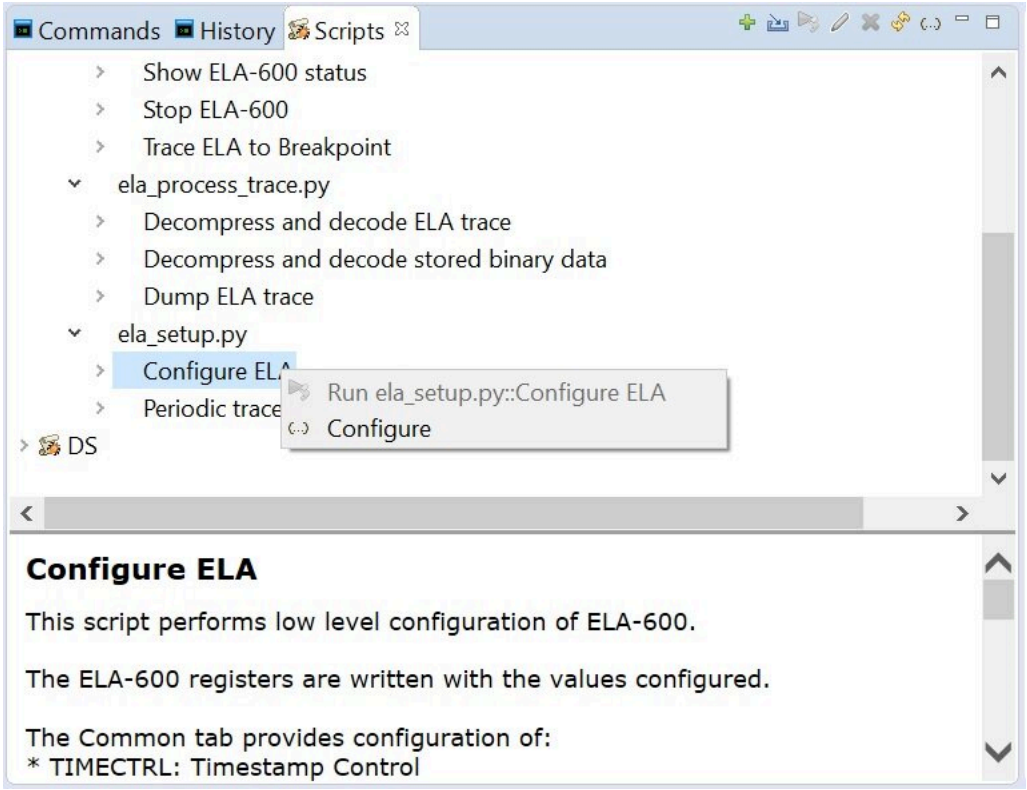
to configure the ELA-600 for our specific debug scenario.


Note

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

1. Connect to the target.
2. Open the GUI ELA-600 configuration utility:
   a. Navigate to: **Scripts window** > **Use case** > **Scripts in DTSLELA-600** > **ela_setup.py** > **Configure**
   b. Right click **Configure ELA** and select **Configure**.

Figure 6-1: Configure ELA



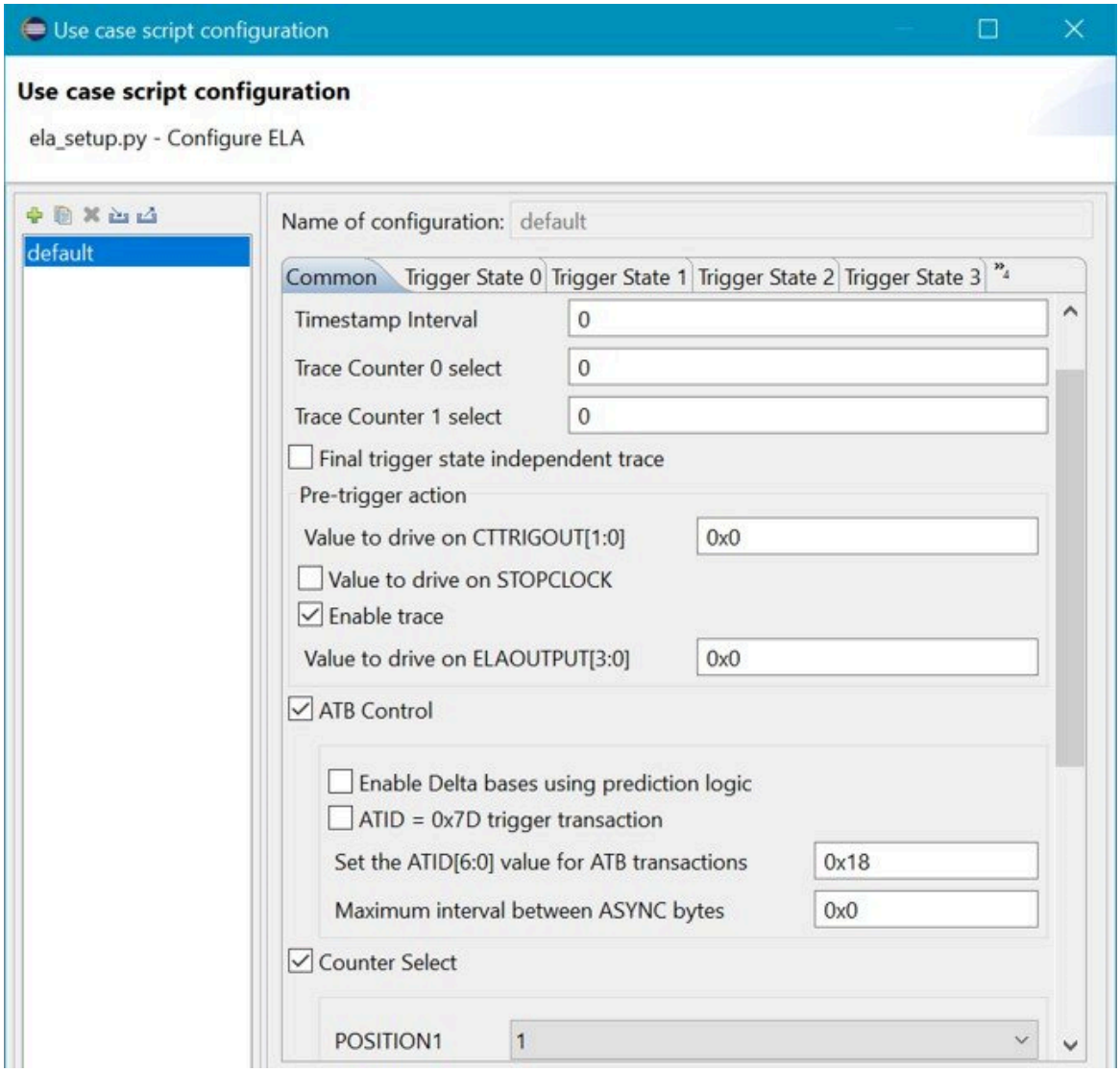
3. Configure the common controls:
   a. Open the **Common** tab.
   b. In the **Pre-trigger action** section, select **Enable trace**.

      This configures the ELA to start tracing when it is enabled - it sets PTACTION.TRACE so that trace becomes active when the ELA-600 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.
   c. In the **ATB Control** section, set **ATID** to `0x18`.

      This sets the ATB trace ID for the ELA-600 to `0x18`. This is used when identifing the trace stream belonging to the ELA-600 if other trace sources are using the ATB.
   d. In the **Counter Select** section, select **Position 1**.

      This makes the trace data capture the counter value for Trigger State 1. The Common tab should look like this when it is finished:

Figure 6-2: Common tab



e. Click **Apply**

4. We now need to configure our first Trigger State:
   a. Open the **Trigger State 0** tab.
   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 `SIGSEL0 == 0x1`. This means that Trigger State 0 will be associated with the trigger signals in Signal Group 0. 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 desired signal locations for other targets, check your IP's corresponding JSON file or documentation.

c. In the **Trigger Control** section:
   1. Set **Signal Comparison (COMP)** to `Equal`.

      This sets the Signal Comparison condition. In this case, we want to trigger when VALID_P1 is '1', Address_P1 is 'B1000000' , and Type_P1 is '1'.
   2. Set **Comparison mode (COMPSEL)** to `1`.

      This enables the Trigger State 0 counters and selects Counter Comparison mode.
   3. Set **Counter source (COUNTSRC)** to `1`.

      This has the Trigger State 0 counter increment on every Trigger Signal Comparison Match.

d. Set the **Next state** value to `0x2`.

   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 move to Trigger State 1 when the Trigger Condition is met.

e. In the Action section:
   1. Set the **CTTRIGOUT[1:0]** value to `0x1`.

      This will have the Trigger State 0 drive CTTRIGOUT[0] upon moving to the next Trigger State. In our case, driving CTTRIGOUT[0] corresponds to a 'halt' signal on the core, so the core will halt when the Trigger State 0 Trigger Condition is met.
   2. Select **Enable trace**.

      This enables Trigger State 0 trace capture.

f. Set **Counter compare** value to `0x3`.

   This sets the Count Compare value to 3, so when there are 3 met comparisons, we will move to Trigger State 1. In our case, the third access to the target address is the 'data corruption' which we are monitoring for.

g. Set both the Signal Mask and Signal Compare fields to the following:
   1. Set the **[31:0]** value to `0x0`.
   2. Set the **[63:32]** value to `0x0`.
   3. Set the **[95:64]** value to `0x200`.
   4. Set the **[127:96]** value to `0x80000162`.

      We need to set Trigger State 0's Signal Compare and Signal Mask values for Signal Group 0 to monitor the VALID_P1, Address_P1, and Type_P1 signals. For the Cortex-A55 + ELA-600 + CCI-500 board, the bit positions are: VALID_P1 at bit position 127, Address_P1 at bit position 114:75, and Type_P1 at bit position 73. You will need to check where these signals are positioned in your IPs corresponding JSON file or documentation.

h. Select **Trace Write Byte Select** and set the **TWBSEL** value to `0x0000FFFF`. This enables trace write for each byte of Signal Group 0 that we want to be trace.The Trigger State 0 tab should look like this when it is finished:

Figure 6-3: Trigger state 0

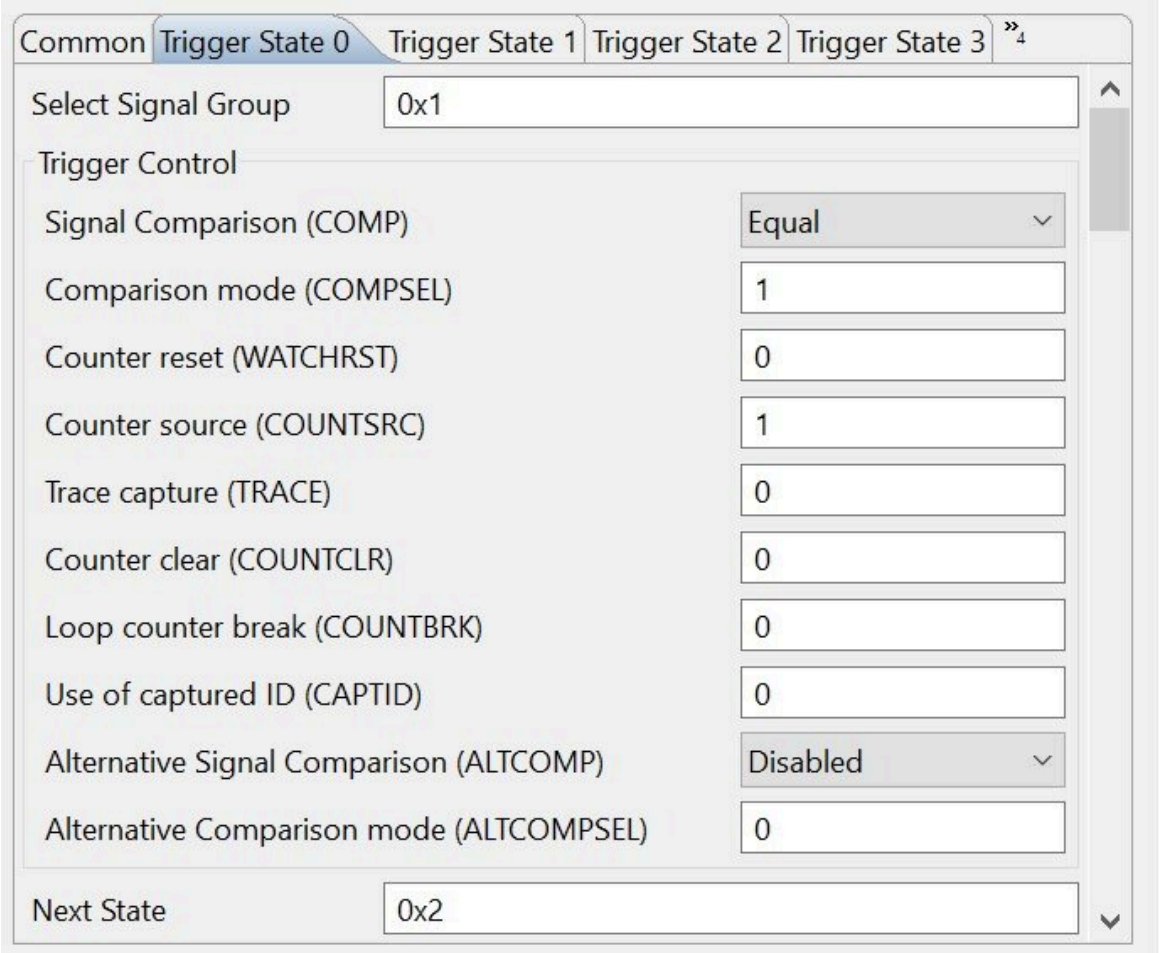


Figure 6-4: Trigger state 0 Action counter

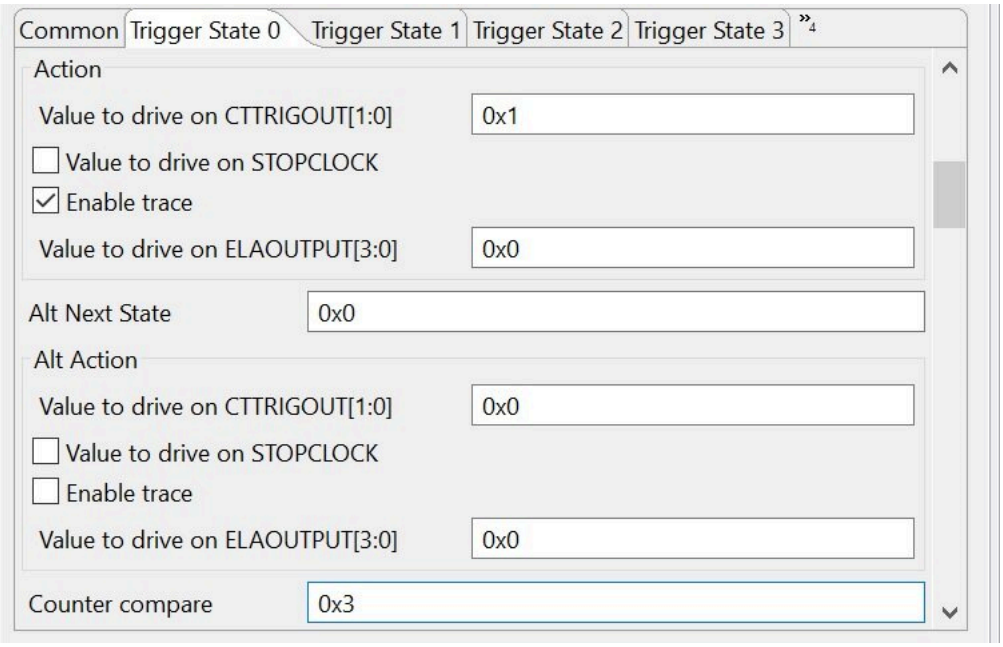


Figure 6-5: Trigger state 0 Signal mask and compare

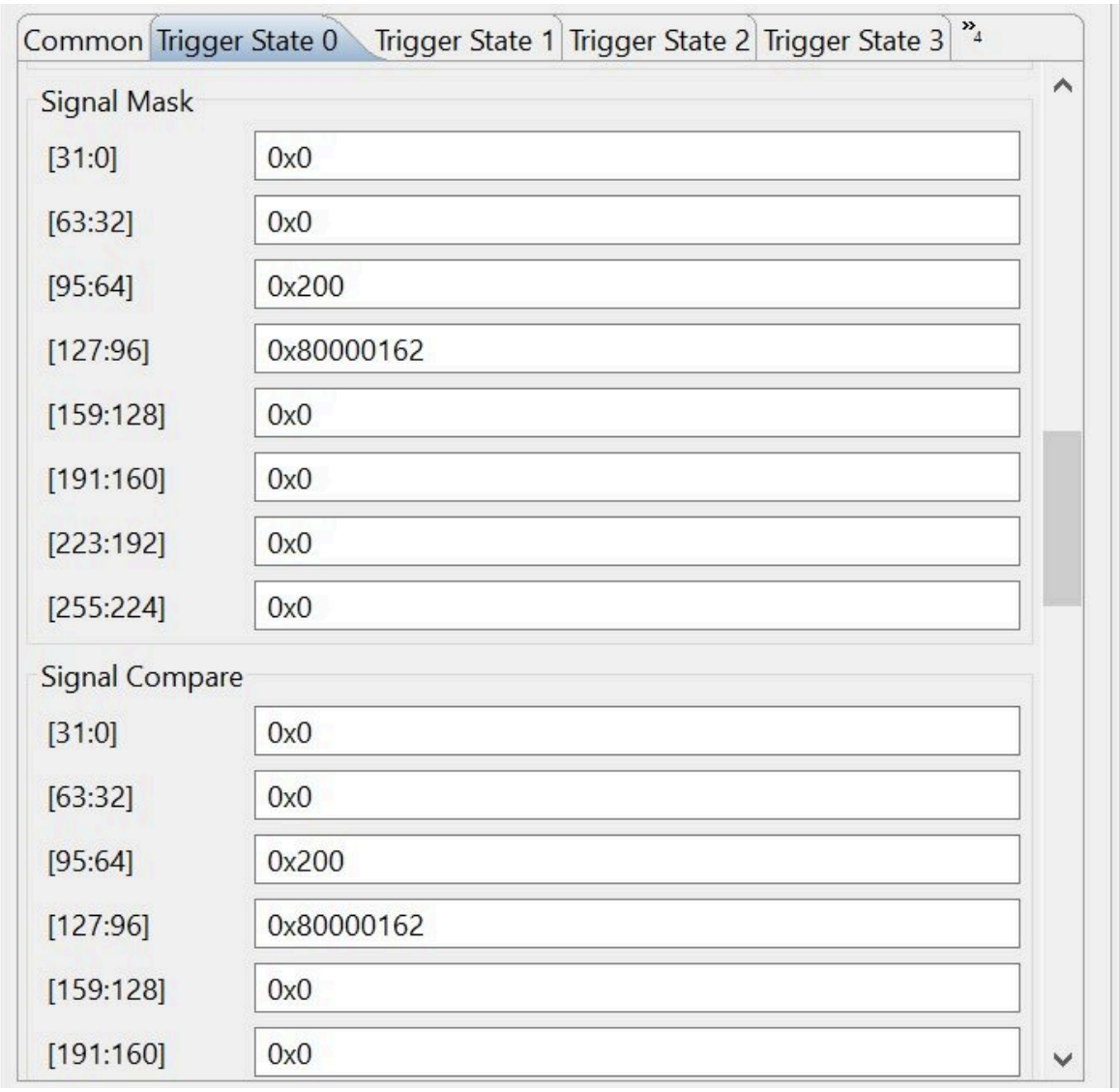
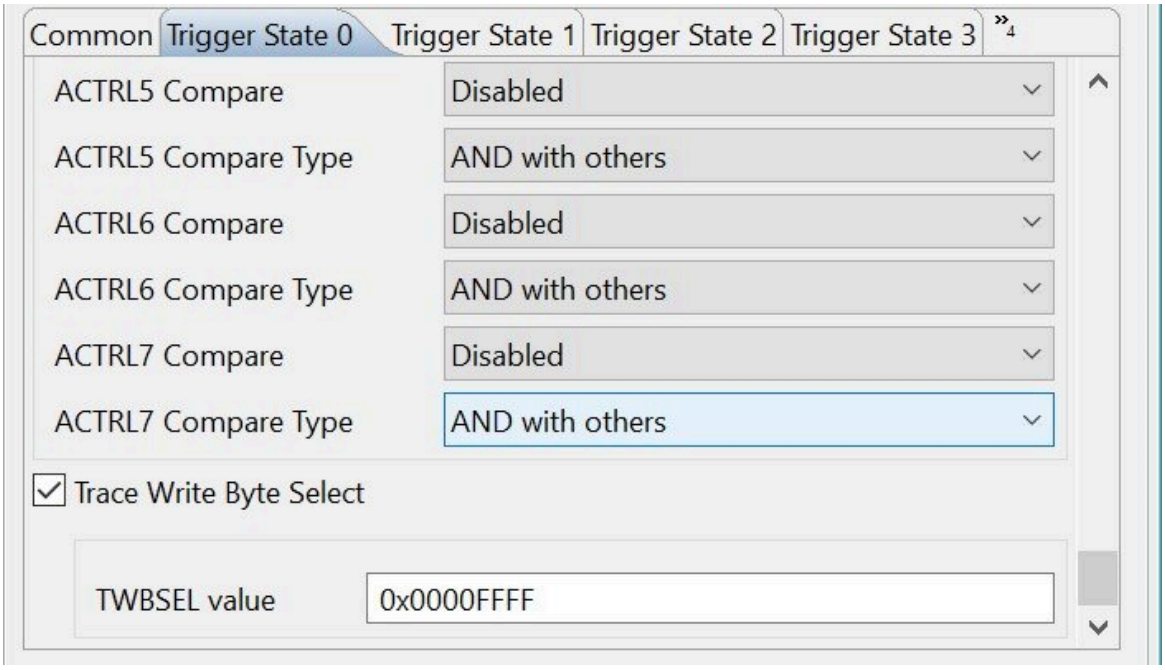


Figure 6-6: Trigger state 0 Trace write byte select



i. Click **Apply**

5. We now need to configure our second Trigger State:
   a. Open the **Trigger State 1** tab.
   b. Set **Select Signal Group** value to `0x0`. This sets it so that the Trigger State will not trigger on any Signal Group. We want to do this in our case as we are using Trigger State 1 to wait for External Trigger input instead.
   c. In the Trigger Control section:
      1. Set **Signal Comparison (COMP)** to `Equal`.

         This sets the Signal Comparison condition. In this case, we want to trigger when an External Trigger Signal from the CTI comes in.
      2. Set **Trace Capture (TRACE)** to `3`.

         This has Trigger State 1 trace for the counter comparison.
   d. Set the **Next state** value to `0x0`.

      Here we set the Next state to '0', so that Trigger State 1 is our final state.
   e. In the **Action** section, select **Enable trace**.

      This enables Trigger State 1 trace capture.
   f. Set **Counter compare** value to `0xFFFFFFFF`.

This sets the Count Compare value to a non-zero value to ensure Trigger State 1 will count. We are using the maximum value allowed as the ultimate count value is unknown in this case.

g. In both the **External Mask** and **External Compare** sections, set **CTTRIGIN[1:0]** to `0x1`.

This will set up a comparison with the external signal associated with the CTIIN[0] bit. In our case, CTIIN[0] is the 'halt' response from the core.

h. Set both the **Signal Mask** and **Signal Compare** fields all to `0x0`.

We need to do this because we do not want to match on any of the signals from any of the Signal Groups.

i. Select **Trace Write Byte Select** and set the **TWBSEL** value to `0x0000FFFF`.

This enables trace write for the data we are interested in.The Trigger State 1 tab should look like this when it is finished:

Figure 6-7: Trigger State 1 Select Signal Group

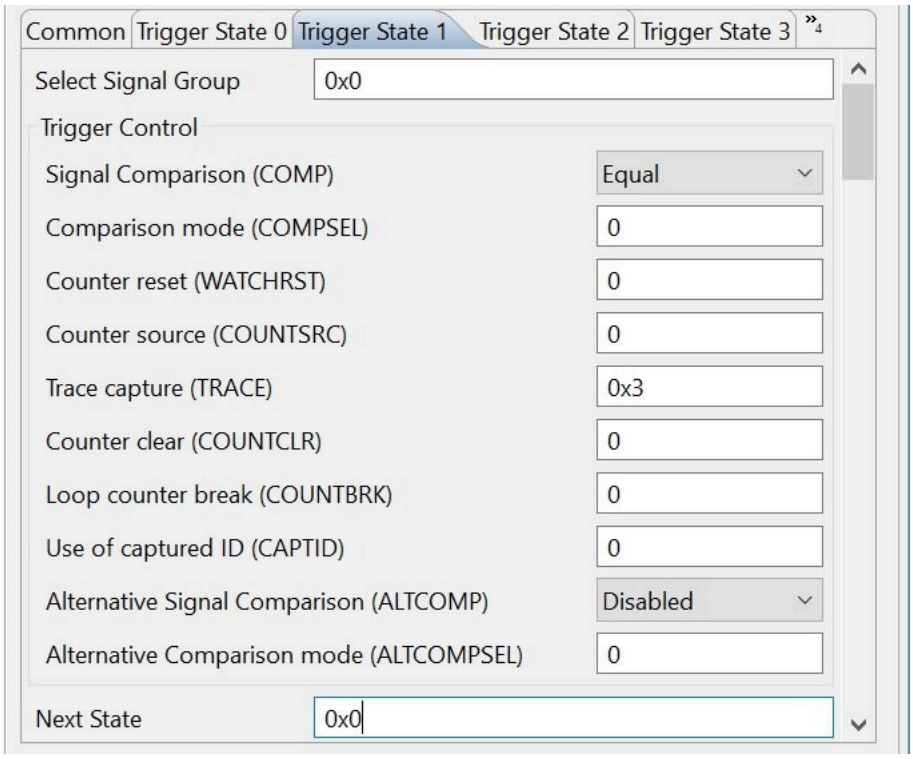


Figure 6-8: Trigger State 1 Action

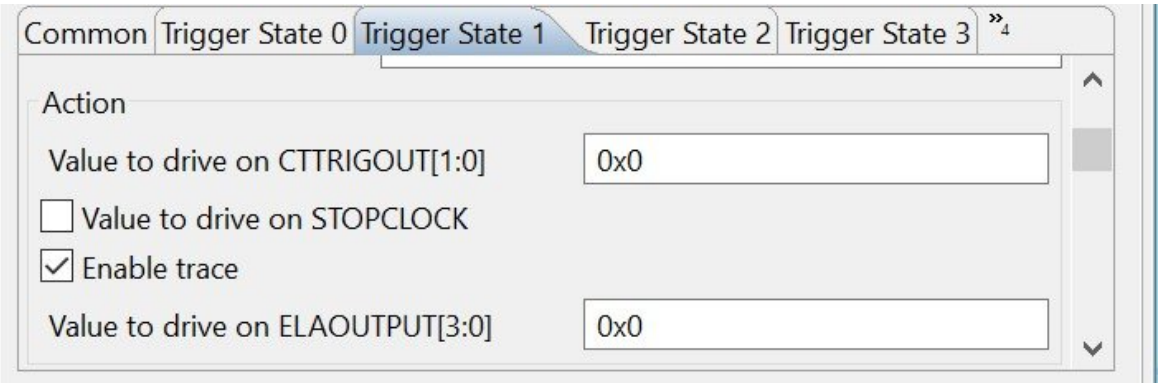


Figure 6-9: Trigger State 1 Mask and Compare

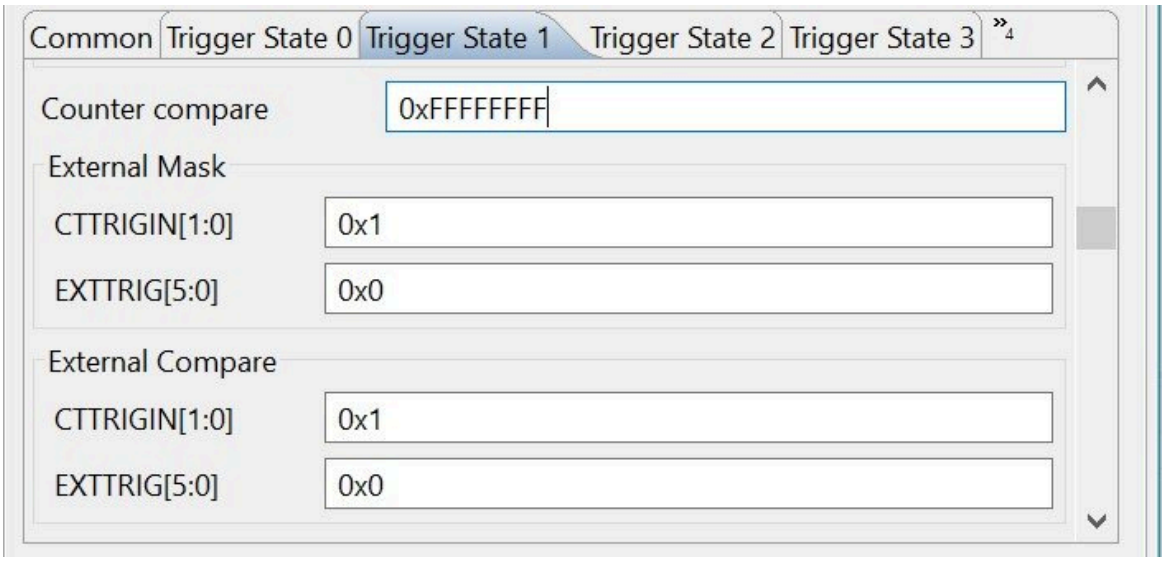
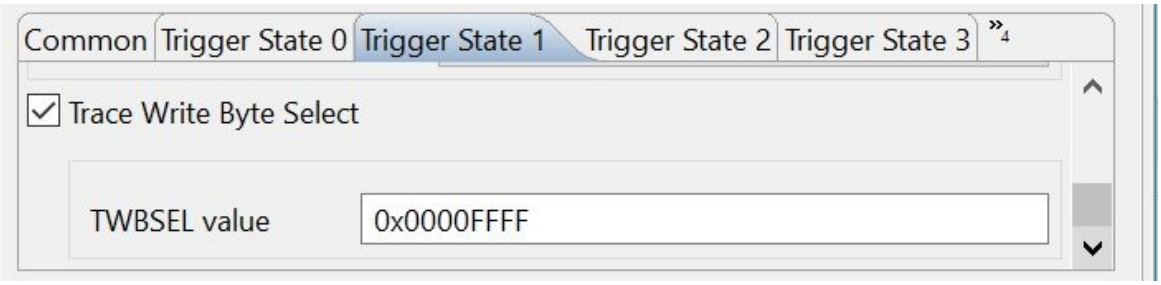


Figure 6-10: Trigger State 1 Trace Write Byte Select



6. Click **Apply** > **OK**.

Alternatively, if you already have a saved ELA-600 configuration, you can import the configuration by clicking the **Import configuration** button in the **ela_setup.py - Configure ELA** view.

An ELA-600 configuration which includes the settings mentioned in this section can be found here (ELA_config.uccfg).

7. Running the ELA use case scripts

The following procedure helps you run the ELA use case scripts:

1. Program the ELA configuration registers:
   a. Navigate to: **Scripts window** > **Use case** > **Scripts in DTSLELA-600** > **ela_setup.py** > **Configure**
   b. Right-click **Configure ELA** and select **Run ela_setup.py::Configure ELA**
2. Setup and enable CTIs for the ELA-600 and the core.

   This will allow the 'halt' request from the Output Action of Trigger State 0 to halt the core and allow Trigger State 1 to see the core halt response. We achieve this by adding some code to the platform configuration's dtsl_config_script.py and running a cti_setup.ds DS-5 script. Both these items can be found in the Eclipse project below.
3. Run the **ELA**:
   a. Navigate to: **Scripts window** > **Use case** > **Scripts in DTSLELA-600** > **ela_control.py** > **Run ELA-600**
   b. 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's trace sink, you will need to:

      1. Right-click **ela_control.py::Run ELA-600** and select **Configure**,
      2. Untick **Start the ETR** when the ELA-600 starts.
      3. Start the trace sink manually or add DTSL lines 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 (sample_stimulus.zip), which contains:
   - Code and an image to run on the target to create the data corruption scenario.
   - An `init.ds` DS-5 script to run as part of the target connection sequence to initialize the Cortex-A55 core.
   - A `cti_setup.ds` DS-5 script to setup the CTI connection between the core and the ELA-600.
   - A `corrupt_mem.ds` DS-5 script to corrupt the memory contents at address `0xB1000000`.
   - A `dtsl_config_script.py` which contains the necessary code to setup the ELA-600 CTI (CSCTI_3).

Result: The target will run and the ELA will be monitoring the input of Signal Group 0 and the External Trigger Signals for the trigger conditions.

8. Capturing ELA trace data

Capturing ELA trace data:

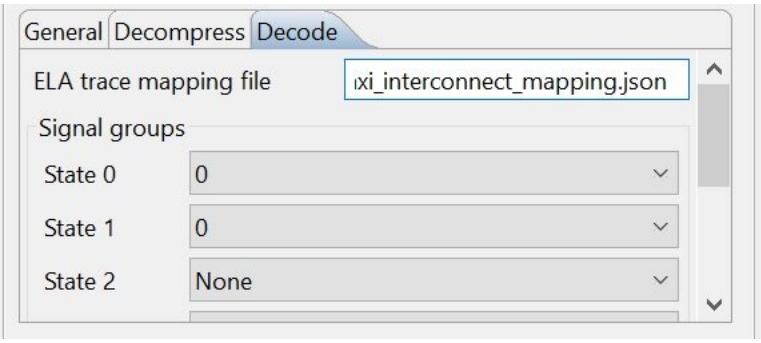
1. In this debug scenario, the core will halt due to the ELA-600 Trigger State 0 programming when the memory at address `0xB1000000` is "corrupted". We can "corrupt" the memory by performing an Debug Access Port (DAP) AXI-AP write to address `0xB1000000` and `0xB1000004` of `0xFFFFFFFF`. Running the corrupt_mem.ds DS-5 script mentioned in the last section will do these writes for us. The core will be running when the corrupt_mem.ds script is executed. The core will halt when the script has completed execution.
2. Stop the ELA:
   a. Navigate to **Scripts window** > **Use case** > **Script in DTSLELA-600** > **ela_control.py** > **Stop ELA-600**
   b. 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's trace sink, you will need to:
      1. Right-click **ela_control.py::Stop ELA-600** and select **Configure**.
      2. Untick **Stop the ETR** when the ELA-600 stops.
      3. Stop the trace sink manually or add DTSL lines to the Stop ELA-600 use case script to stop the trace sink.
3. Dump and decode the ELA trace:
   a. Navigate to: **Scripts window** > **Use case** > **Scripts in DTSLELA-600** > **ela_process_trace.py** > **Decompress and decode ELA trace**

   The Decode trace data script requires the corresponding JSON file called axi_interconnect_mapping.json. The location of this file can be found in the DTSLELA-600 directory.

   b. Right click on **Decompress and decode ELA trace** and select **Configure**
      1. Under the General tab:
         a. Select **Decompress and decode trace**
         b. Select **Output to screen** or **Output to file...** and then provide a file path and name if the trace data is large in size
      2. Under the **Decode** tab:
         a. Set **ELA trace mapping file** to `DTSLELA-600 axi_interconnect_mapping.json` file
         b. Under the **Signal groups** section, set **State 0** and **State 1** to `0`. The Decode tab should look like this when it is finished:

Figure 8-1: Decode tab



c. Right click on **Decompress and decode ELA trace** and select **Run ela_process_trace.py::Decompress and decode ELA trace**.

If you select **Output to screen** in the **General** tab, the decompressed and decoded trace data will appear in the DS-5 Commands view.

If you select **Output to file...** in the **General** tab, the decompressed and decoded trace will appear in text format in the file you specify.

9. Analyzing the ELA trace data

As a result of our work above, the ELA traces 3 transactions to address `0xB1000000` and some counter values and outputs it 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` T he CNTSEL[0] counter value is the time between Trigger State 0 issuing a halt request to the core because of the “data corruption” and when the core responded with a reciprocal signal.

The snippet of our trace capture below shows the decompressed and decoded trace data for the above 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: 0xA0300162000002C81C00400000602675
```

```
 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
```

10. Resources

Here are some resources related to material in this guide:

- The Arm CoreSight ELA-600 Embedded Logic Analyzer Product Page
- ELA-600 Technical Reference Manual (TRM)
- Application Note - Arm CoreSight ELA-600 Version 1.0
- Support for ELA-600 (a DS-5 Development Studio Video)