arm

Using the ELA-600 with Arm DS

Version 1.0

Using the ELA-600 with Arm DS

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	29 April 2021	Non-Confidential	First release

Proprietary Notice

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 using the ELA-600 with Arm DS

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.

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.

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.

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.
  - 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.
  - 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.

3. Importing the DTSL ELA-600 use case scripts

The following instructions shows you how to import the ELA-600 DTSL 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**.

   Figure 3-1: Import Examples and Programming Libraries
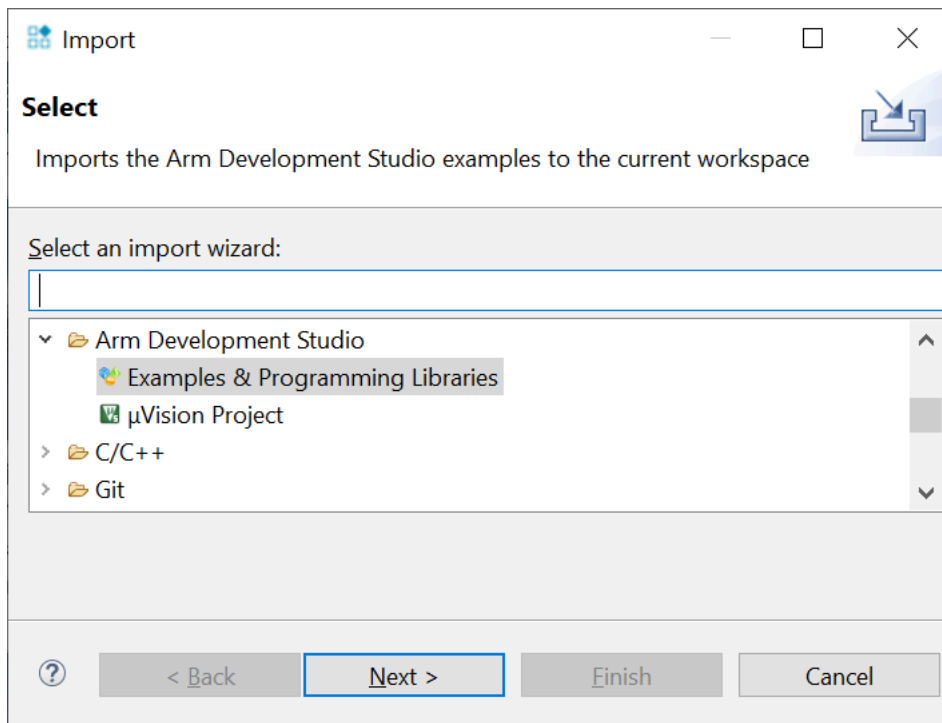
5. Expand **Examples** > **Debug and Trace Services Layer (DTSL)**.

6. Select **DTSLELA-600**.

Figure 3-2: Import DTSLELA-600 project
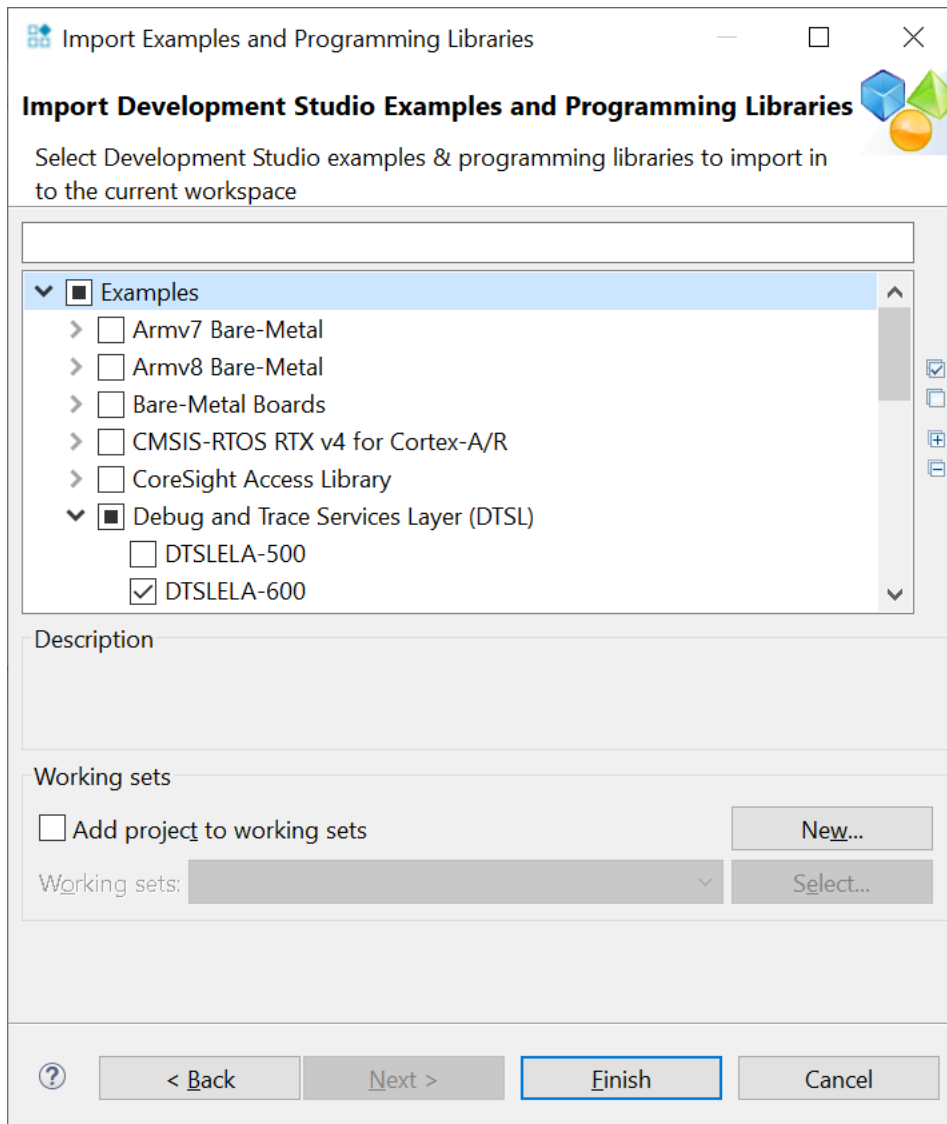
7. Click **Finish**.

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.

Figure 4-1: SDF file Device Table list

Figure 4-2: SDF file Component Connections list
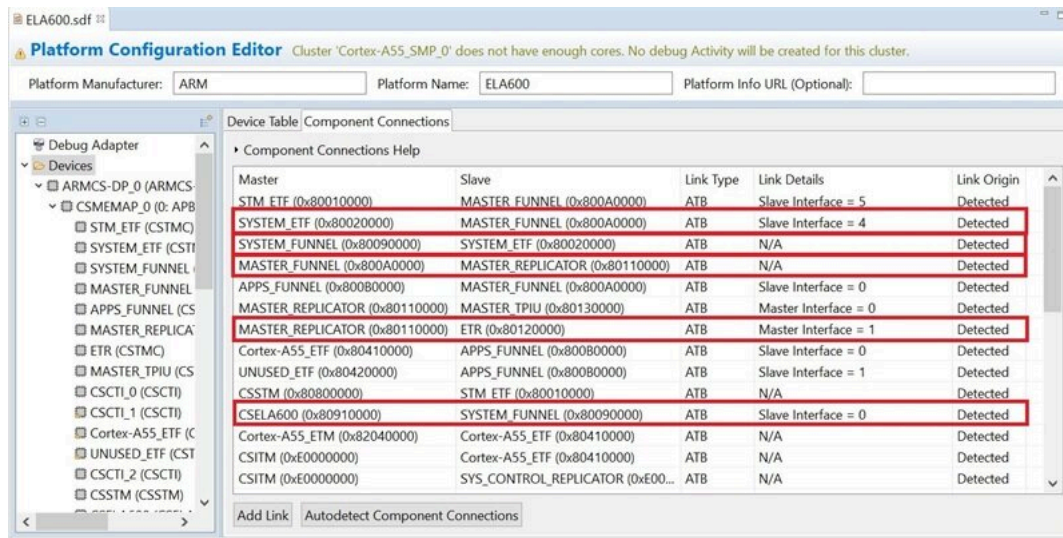
All the Arm DS DTSL ELA-600 use case scripts assume the ELA-600 is named "CSELA600" in the platform configuration SDF file. To use your platform configuration with the Arm DS use case scripts, you either must:

Note

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

Figure 4-3: SDF File Device Name change
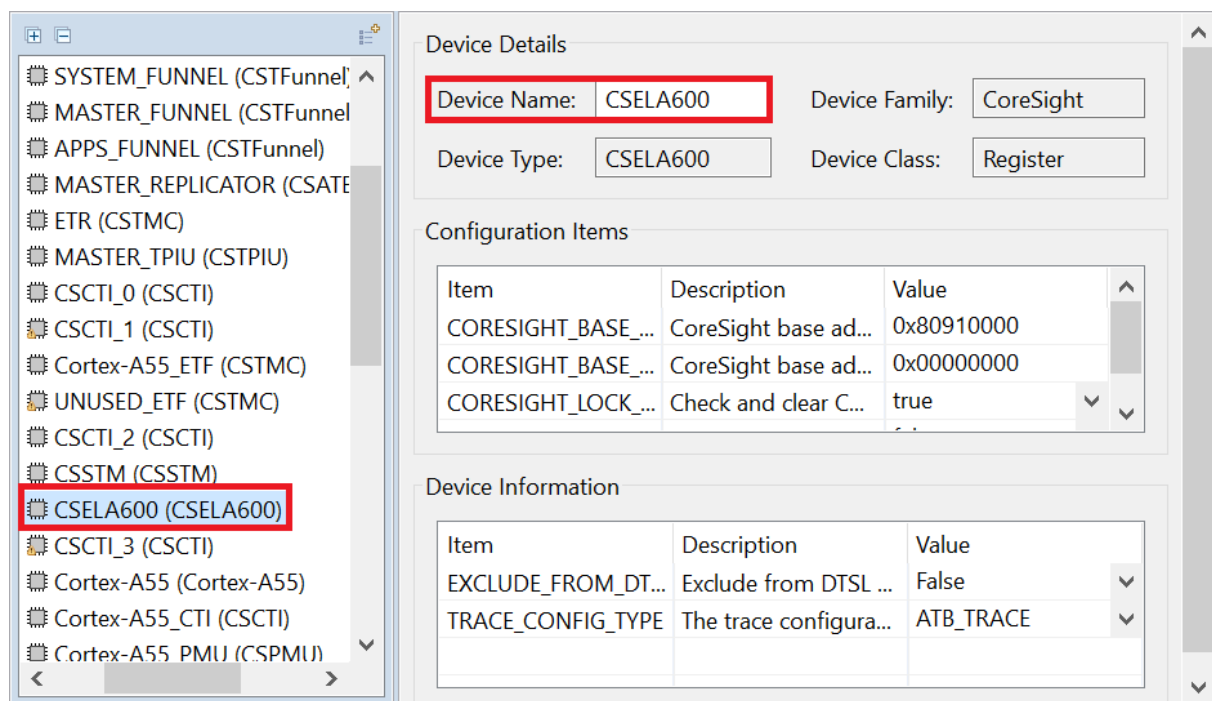
- Manually configure the **ELA-600 device name** field for each use case script to the **Device Name** specified in the SDF file. Save the use case script changes by clicking **Apply** and **OK**. As an example of one of the scripts to change, the following shows the **ELA-600 device name** field in the **ela_setup.py – Configure ELA** use case script.

Figure 4-4: Use case script ELA-600 device name change
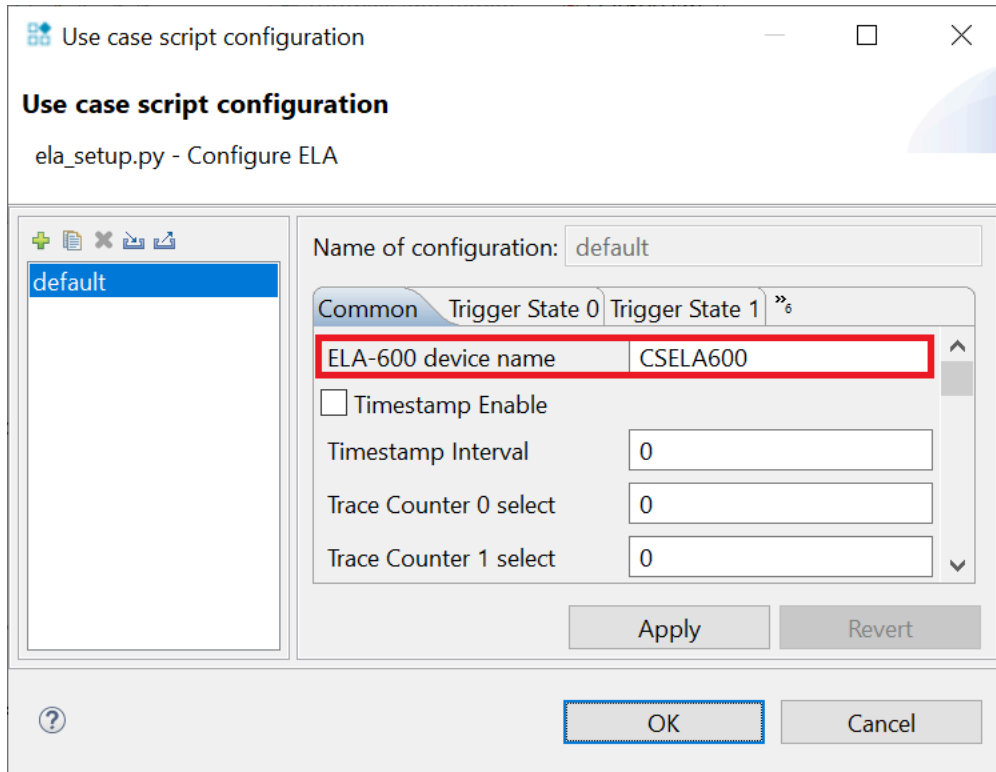
5. Setting up the DTSL options

To work with the DTSL ELA-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**.

Figure 5-1: DTSL enable ELA
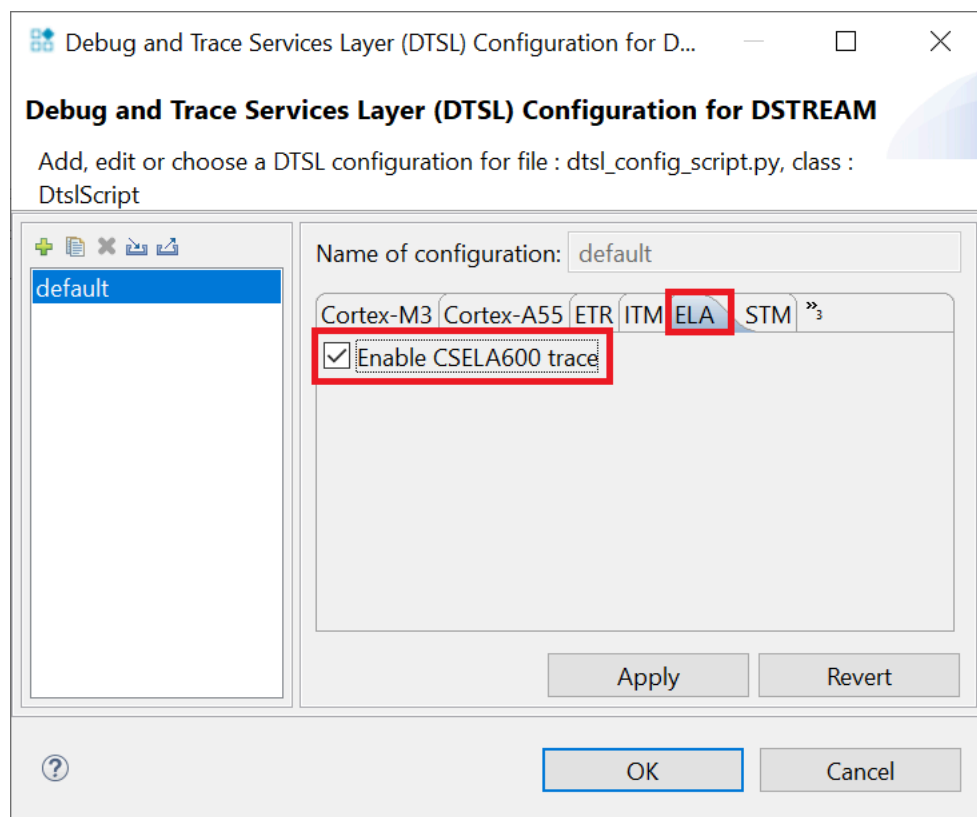
2. To enable trace data capture to the ETR, in the **Trace Capture** tab, set **Trace capture method** to **System Memory Trace Buffer (ETR/ETR)**.

Figure 5-2: DTSL enable Trace Capture
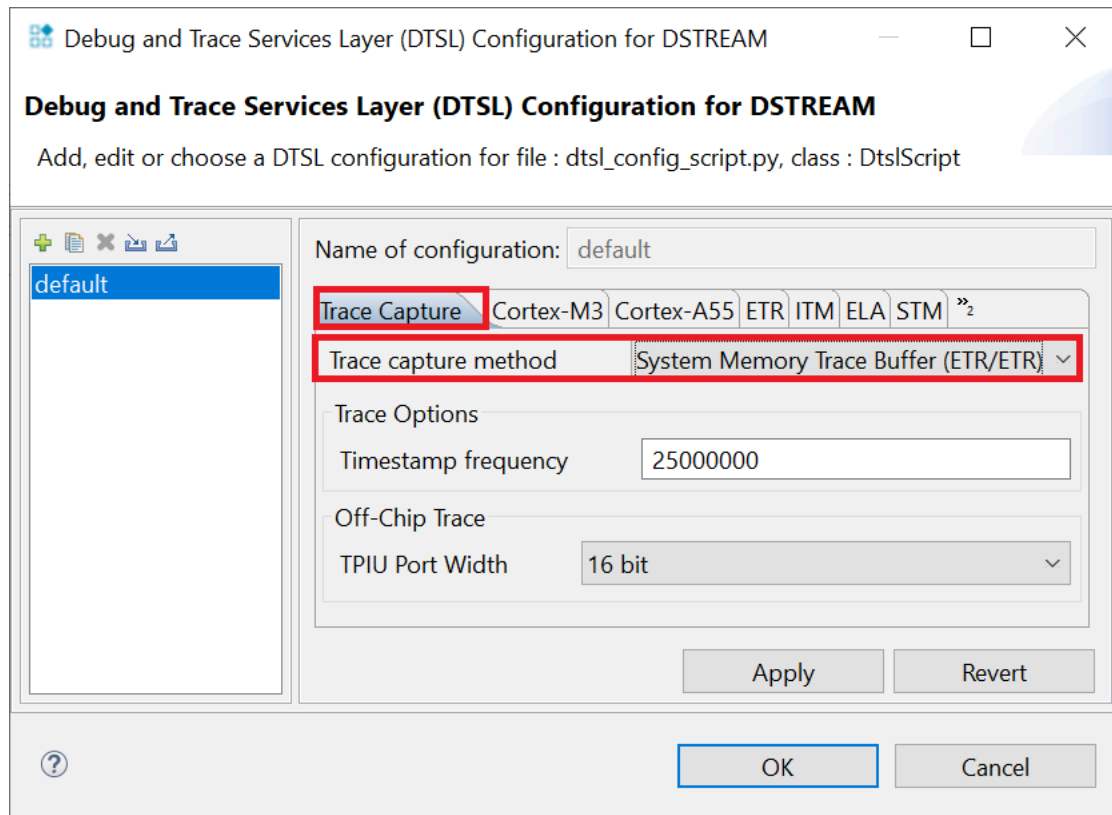
3. To set up the ETR, in the **ETR** tab:

   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.

Figure 5-3: DTSL enable ETR
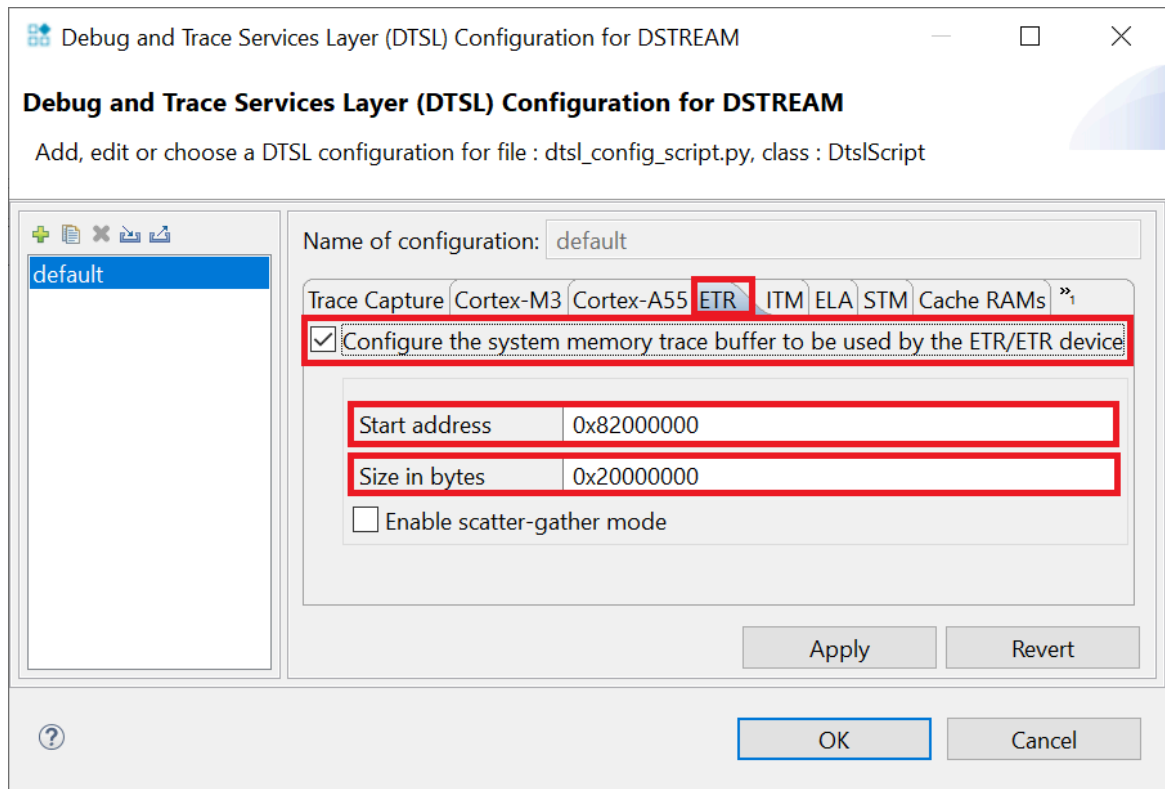
6. Configuring the ELA-600 use case scripts

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...**.

   Figure 6-1: Add Use Case Script Directory... to Scripts view
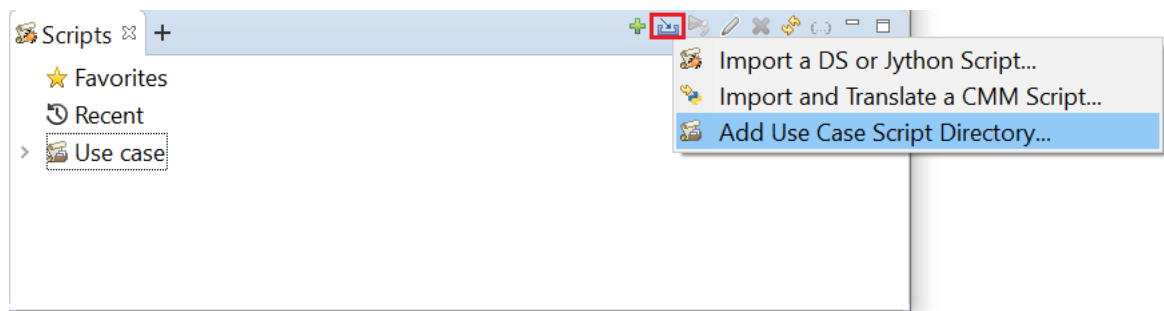
4. In the **Select Folder** dialog, browse to the DTSLELA-600 project in your Arm DS Workspace and click **Select Folder**.

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

Figure 6-2: After use case scripts are added in Scripts view
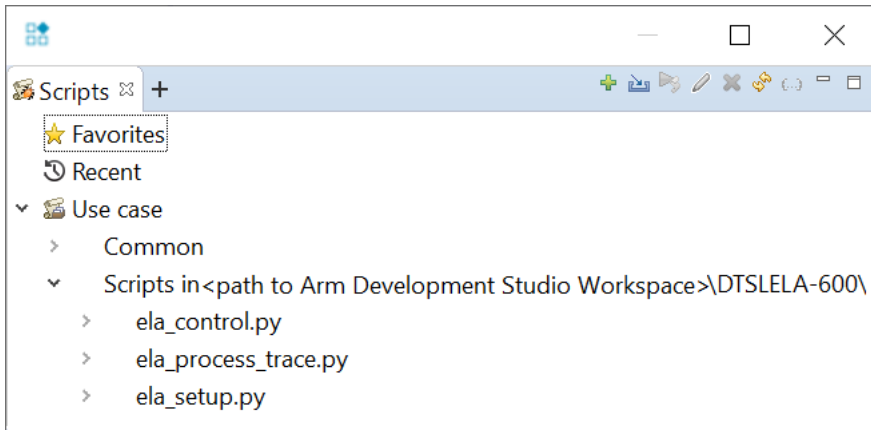
For this demonstration, we use the GUI ELA-600 Configuration Utility provided by the Arm DS use case script at

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...** .

Figure 6-3: Configure... the ela_setup.py Configure ELA use case script
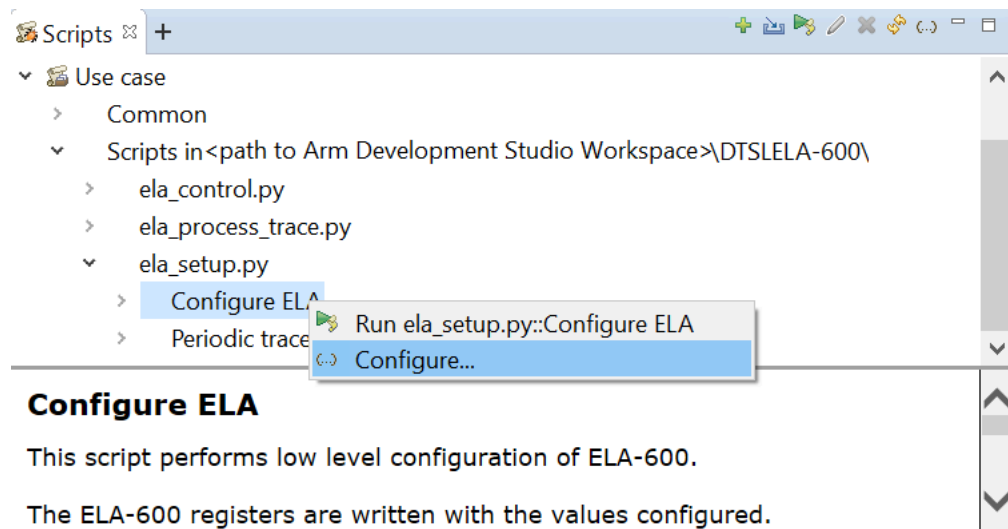
We now configure the Common registers:

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:

Figure 6-4: Common tab of Configure ELA view
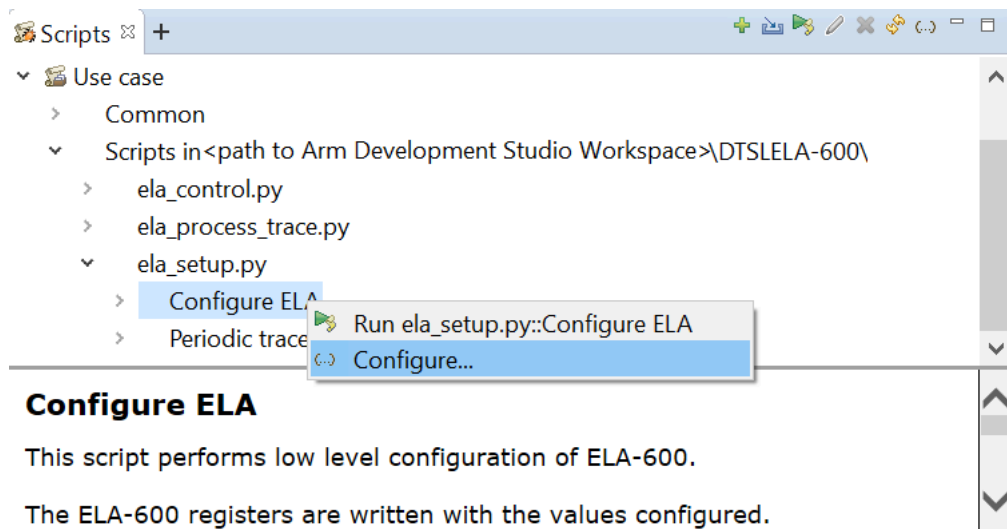
5. Click **Apply**.

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:

Figure 6-5: Trigger State 0 Selection and Control of Configure ELA view

Figure 6-6: Trigger State 0 Action and Counter of Configure ELA

Figure 6-7: Trigger State 0 Signal Mask and Compare of Configure ELA view

Figure 6-8: Trigger State 0 Trace Write Byte Select of Configure ELA view

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:

**Figure 6-9: Trigger State 1 Select and Control of Configure ELA view**

Figure 6-10: Trigger State 1 Action of Configure ELA view

Figure 6-11: Trigger State 1 Counter and External CTTRIGIN of Configure ELA view

Figure 6-12: Trigger State 1 Trace Write Byte Select of Configure ELA

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 (ELA_config.uccfg).

7. Running the ELA use case scripts

The following instructions shows you how to run the ELA use case scripts:

1. Program the ELA configuration registers:

    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 **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:

    a. Navigate to: **Scripts** view > **Use case** > **Scripts in <path to Arm Development Studio Workspace>\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 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

The following instructions shows you how to capture 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:

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

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

   c. 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**.

   a. Under the **General** tab, select **Decompress and decode trace**.

   b. 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.

   c. 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.

   ---

   d. 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:

Figure 8-1: Decode tab settings
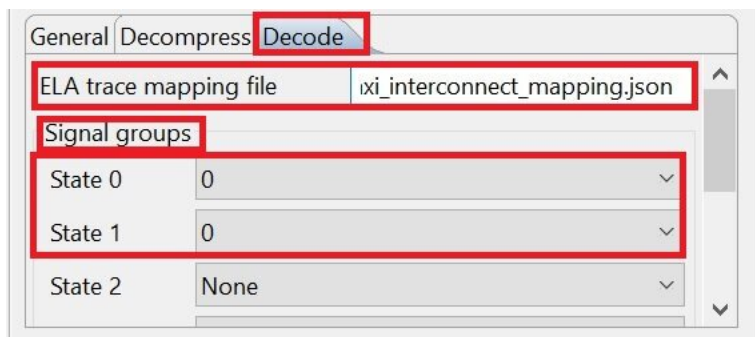
e. Click **Apply** and **OK**.

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: 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
  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: 0x000000030000000030000000030000000B
```

```
CNTSEL[0] : 32'hb
CNTSEL[1] : 32'h3
CNTSEL[2] : 32'h3
CNTSEL[3] : 32'h3
```

10. Resources

The following resources are related to this guide:

- 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)