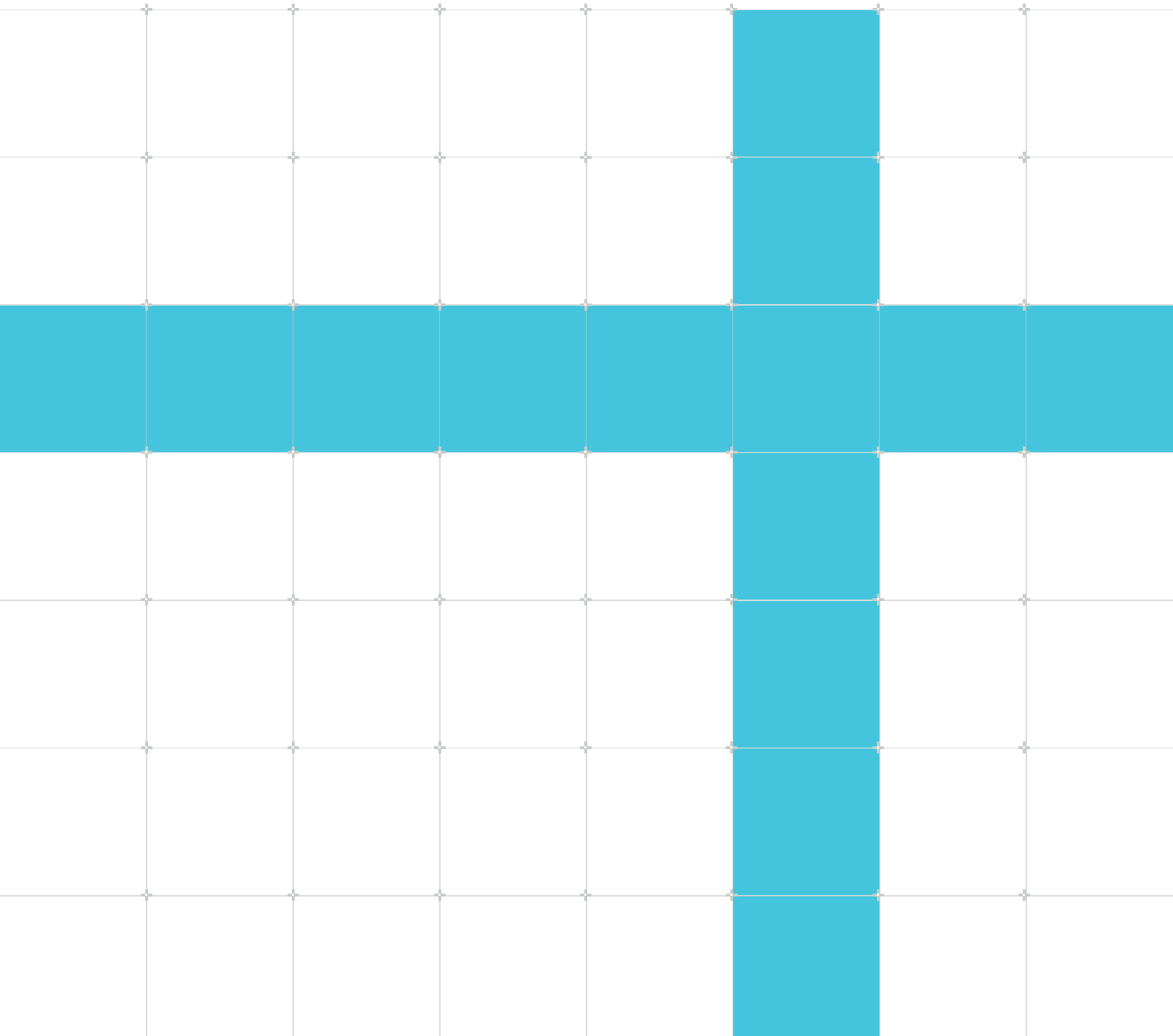
Using the ELA-600 with Arm DS

Version 3.0

guide

Issue 01
102592_3_01_en



Using the ELA-600 with Arm DS guide

Release information

Document history

Issue	Date	Confidentiality	Change
0300-01	12 February 2026	Non-Confidential	Added information for Arm Development Studio 2025.0-2
0200-01	30 July 2025	Non-Confidential	Second release
0100-01	29 April 2021	Non-Confidential	First release

Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Confidentiality Status

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 (ELA-600) 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

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

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

ELA-600 provides a superset of the functionality of the Arm CoreSight ELA-500 Embedded Logic Analyzer (ELA-500). The largest functional additions are acquiring trace data over the Advanced Trace Bus (ATB) and having potentially 8 Trigger States instead of 5.

Read more about the ELA-600 in the Resources section of this tutorial.

The problem

Memory data corruption occurs when on-chip operations like the following happen:

- Unintended code execution
- Stack or heap leakage
- Externally like through malicious code.

To diagnose memory data corruption, 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 example scenario:

- Memory accesses to a specific address of `0xB1000000` are tracked 3 times.
- The ELA-600 monitors the memory accesses and waits 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 that is available in the Application Note - Arm CoreSight ELA-600 Version 1.0.

The solution

In this scenario, to trace the external bus transactions made by the processor, an ELA-600 is used. An example target with a Cortex-A55, ELA-600, and Cache Coherent Interconnect 500 (CCI-500) is used for the scenario. This guide uses the ELA-600 DTSL use case scripts shipped with Arm Development Studio (Arm DS).

2. Before you begin

If you have not already, install Arm Development Studio (Arm DS).

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

- 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.
- A platform configuration that:
  - Lists the relevant ELA trace component or components.
  - If ELA-600 trace is captured to the ATB interface, a list of the component connections and the mapping between the ELA and its trace sink.
  - If using Cross Trigger Interfaces (CTIs), a list of the CTIs and their connections.

3. Importing and adding the DTSL ELA-600 use case scripts

Depending on your Arm DS version, you might need to import and add the DTSL ELA-600 use case scripts to Arm DS.

For Arm DS 2025.0-2 or later, you do not need to import nor add the ELA-600 use case scripts. The ELA-600 use case scripts are included as part of the configuration database (configdb) shipped with Arm DS. This inclusion means that, after connecting to any target, the ELA-600 use case scripts are available in the **Scripts** view. If the **Scripts** view is not present in your Arm DS Workspace, open the **Scripts** view by selecting **Window** > **Show View** > **Scripts**.


Note

To see the ELA-600 use case scripts, you must have connected to 1 target at least once.

Importing and adding the DTSL ELA-600 use case scripts for older Arm DS versions

For Arm DS versions earlier than 2025.0-2, you must import the DTSLELA-600 project that contains the scripts and add the scripts to the **Scripts** view.

Import the ELA-600 DTSL use case scripts using the following steps:

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**.
5. Expand **Examples** > **Debug and Trace Services Layer (DTSL)**.
6. Select **DTSLELA-600**.
7. Click **Finish**.

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

Add the **DTSLELA-600** project use case scripts to the **Scripts** using the following steps:

1. Connect to a target with Arm DS.

   Learn how to connect to a target with Arm DS, read the "Debugging code" section of the Arm Development Studio Getting Started 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...**.

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

Result: In the **Scripts** view, the DTSLELA-600 use case scripts appear under **Use Case** > **DTSLELA-600**.

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 1 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.

The following shows the necessary platform configuration items for an example target with an ELA-600 using a CTI and tracing over ATB:

Figure 4-1: SDF file Device Table list

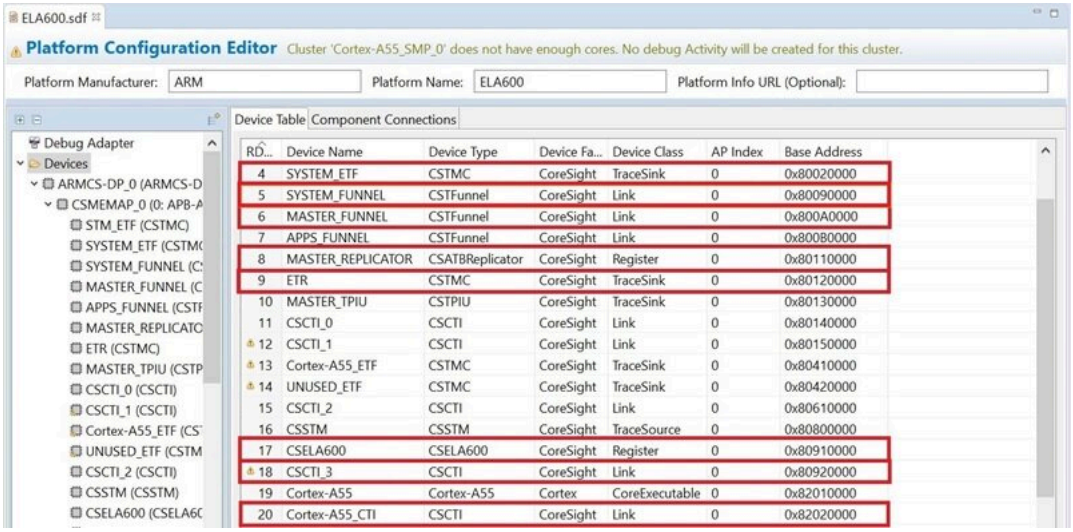
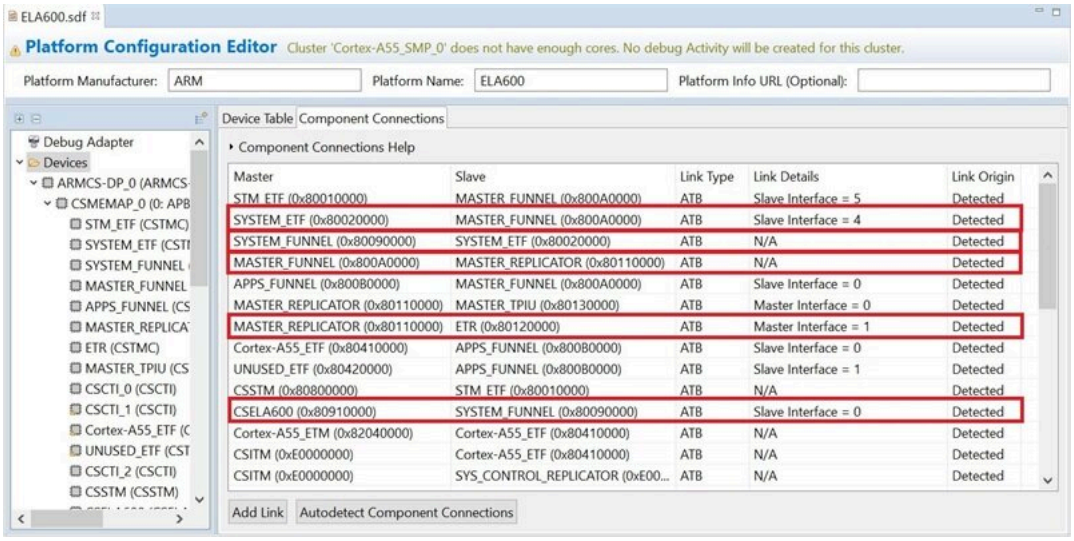


Figure 4-2: SDF file Component Connections list



Note

In platform configurations, Arm DS does not support having a component connection between an ELA-600 and a CTI.

If you are using an Arm DS version earlier than 2025.1, all the Arm DS DTSL ELA-600 use case scripts assume the ELA-600 is named `CSELA600` in the SDF file in the platform configuration. To use your platform configuration with the Arm DS use case scripts, you must do 1 of the following:

- If you are only using 1 ELA, in the SDF file, change the ELA-600 component **Device Name** to `CSELA600`. Save the change with **File** > **Save**.
- If are using more than 1 ELA, manually configure the **ELA-600 device name** field for each use case script to the **Device Name** specified for the ELA in the SDF file. Save the use case script changes by clicking **Apply** and **OK**.

5. Configuring the ELA-600 use case scripts

To configure the ELA-600, you can use the configuration dialog provided by the **DTSLELA-600 ela_setup.py** use case script. This dialog 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 configure the ELA-600 for your specific debug scenario, use **ela_setup.py**. In this example, we configure the ELA-600 to help debug the data corruption scenario explained in The problem section of this guide.

Note

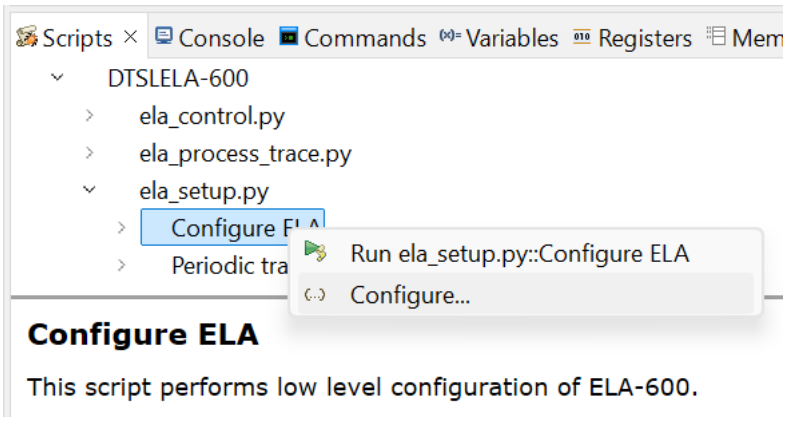
Before configuring the ELA-600, you must connect Arm DS to the target.

Do the following to open the configuration dialog:

1. Navigate to **Scripts** view > **Use case** > **DTSLELA-600** > **ela_setup.py** > **Configure ELA**.
2. Right-click **Configure ELA** and select **Configure**.

The following image shows how to get to **Configure**:

Figure 5-1: Configure the ela_setup.py Configure ELA use case script



If using Arm DS 2025.0-2 or later, the following 4 new fields are available in the **Common** tab:

- Dropdown option available from **ELA-600 device name**.
- **Pre-run Script**
- **Signal Mapping File**
- **Signal mapping directory**

You can select which ELA you are configuring using the dropdown option available from **ELA-600 device name**. The ELA list that is shown in the dropdown is based on the ELAs that are listed in the SDF file in the platform configuration. The icons and space on the left side of the dialog allows you to easily manage your ELA use case script configurations. To learn more about ELA configurations, read the "Manage ELA configurations" section of the Arm Development Studio User Guide.

With **Pre-run Script**, you can optionally select an Arm DS script to run before the ELA is configured. To learn about Arm DS scripts, read the "Debugging with Scripts" section of the Arm Development Studio User Guide.

So the Trigger State tabs show signal names associated with the ELA signal mapping for a JSON file, select 1 of the following options:

- **Signal Mapping File**
- **Signal mapping directory**

To use **Signal mapping directory**, you must select the ELA you are configuring from the **ELA-600 device name** dropdown list.

Contact your IP designer for the JSON file for your specific ELA signal mapping.

Note

If you choose to not set either **Signal Mapping File** or **Signal mapping directory**, select **Signal Mapping File** and delete the JSON path and file shown in the field. You must delete the path and file shown so no signal mapping is shown in the Trigger State tabs.

Now configure the common registers:

1. Open the **Common** tab.
2. Ensure **ELA-600 device name** is set to **CSELA600**.
3. If using Arm DS 2025.0-2 or later, select **Signal Mapping File** or **Signal mapping directory** and set the file path or directory to your signal mapping JSON file. For this example, leave the default JSON file path of *axi_interconnect_mapping.json*.
4. In the **Pre-trigger action** section, select **Enable trace**.

   When the ELA-600 is enabled, this setting configures the ELA to start tracing. Configuring **Pre-trigger action** sets TRACE[3] 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.
5. For Arm DS versions earlier than 2025.0-2, tick **ATB Control** and set **ATID[6:0] value for ATB transactions** to `0x18`.

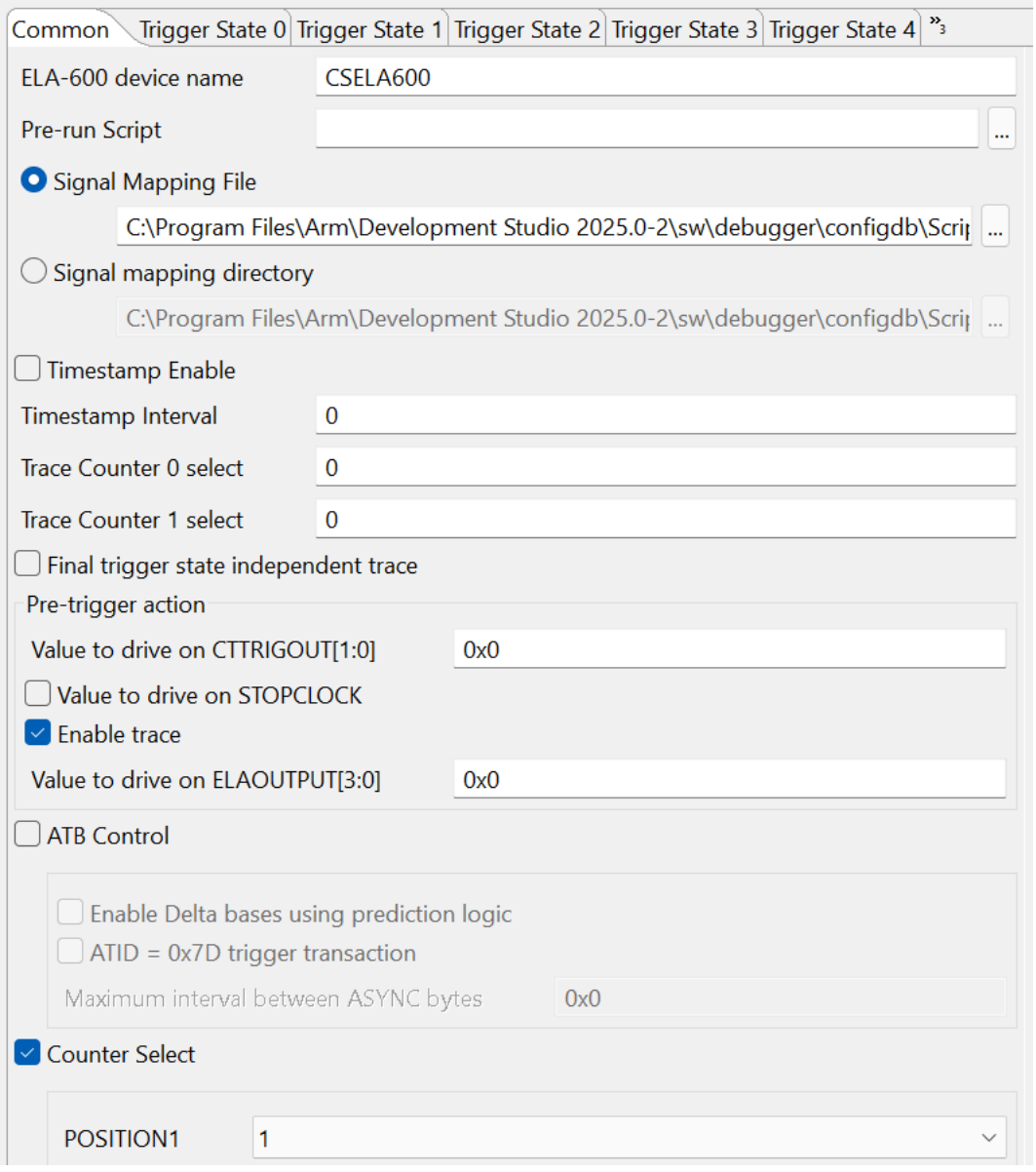
   This setting configures the ATB trace ID for the ELA-600. Configuring **ATID[6:0] value for ATB transactions** sets ATID_VALUE[14:8] of the ATB Control register (ATBCTRL). If other trace sources are using the ATB, this value is used to identify the trace stream belonging to the ELA-600.
6. Tick **Counter Select** and set **Position 1** to **1**.

This setting makes the trace data capture the counter value for Trigger State 1. Configuring **Position 1** sets POSITION1[2:0] of the Counter Select register (CNTSEL).

7. Click **Apply**.

When the previous steps are completed, the **Common** tab shows the following for Arm DS 2025.0-2 and later:

Figure 5-2: Common tab of Configure ELA view



Now to 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. Configuring **Select Signal Group** sets SIGSEL0[11:0] of the Signal Select 0 register (SIGSEL0). The ELA-600 uses a one-hot encoding for the Signal Group in the Signal Select registers.

   In our example system, the VALID_P1, Address_P1, and Type_P1 signals reside in Signal Group 0. To locate the wanted signal locations for other targets, check the JSON file or documentation for your IP.

3. In the **Trigger Control** section:
   a. Set **Signal Comparison (COMP)** to **Equal**.

      This setting configures the Signal Comparison condition for the Trigger State. Configuring **Signal Comparison (COMP)** sets COMP[2:0] 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. Configuring **Comparison mode (COMPSEL)** sets COMPSEL[3] 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. Configuring **Counter source (COUNTSRC)** sets COUNTSRC[5] of the TRIGCTRL0 register.

4. Set **Next State** to **Trigger State 1** or `0x2`.

   Here we set the Next State. This state is the ELA state we enter when we meet the Trigger Condition. Configuring **Next State** 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 **Value to drive on CTTRIGOUT[1:0]** to `0x1`.

      This setting has Trigger State 0 drive CTTRIGOUT[0] when moving to the next Trigger State. Configuring **CTTRIGOUT[1:0]** sets CTTRIGOUT[1:0] 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. Ticking **Enable trace** sets TRACE[3] of the ACTION0 register.

6. Set **Counter compare** to `0x3`.

   This setting configures COUNTCOMP0[31:0] 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. 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. Configuring **TWBSEL** sets TRACE_BYTE[15:0] of the Trace Write Byte Select 0 register (TWBSEL0).

8. We must configure SIGMASK[255:0] of the Signal Mask 0 registers (SIGMASK0) and SIGCOMP[255:0] of the Signal Compare 0 registers (SIGCOMP0) to monitor the VALID_P1, Address_P1, and Type_P1 signals. For the example system, the bit positions of these signals 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.

   If using Arm DS 2025.0-2 or later, in the signal mapping table, set the following signal names to these values:

   - For **Type_P1[3:0]**, set **Mask** to `0x1` and **Comparison Value** to **Read Once**.
   - For **Address_P1[42:0]**, set **Mask** to `0x00B1000000` and **Comparison Value** to `0x00B1000000`.
   - For **VALID_P1**, set **Mask** to `0x1` and **Comparison Value** to `0x1`.

   For Arm DS versions earlier than 2025.0-2, 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`.

9. Click Apply.

When the previous steps are completed, the **Trigger State 0** tab looks like the following for Arm DS 2025.0-2 and later:

Figure 5-3: First half of Trigger State 0 of Configure ELA view

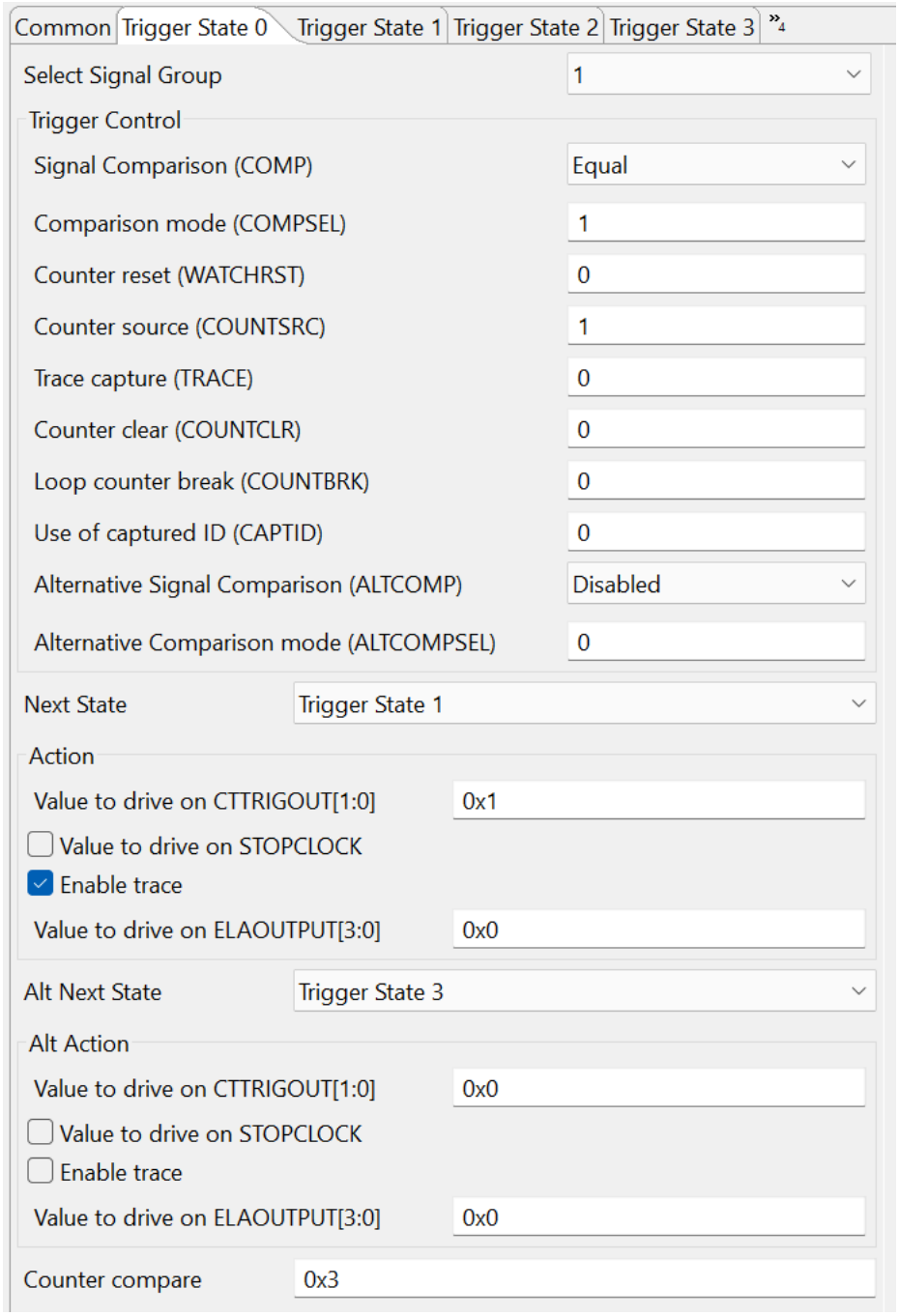
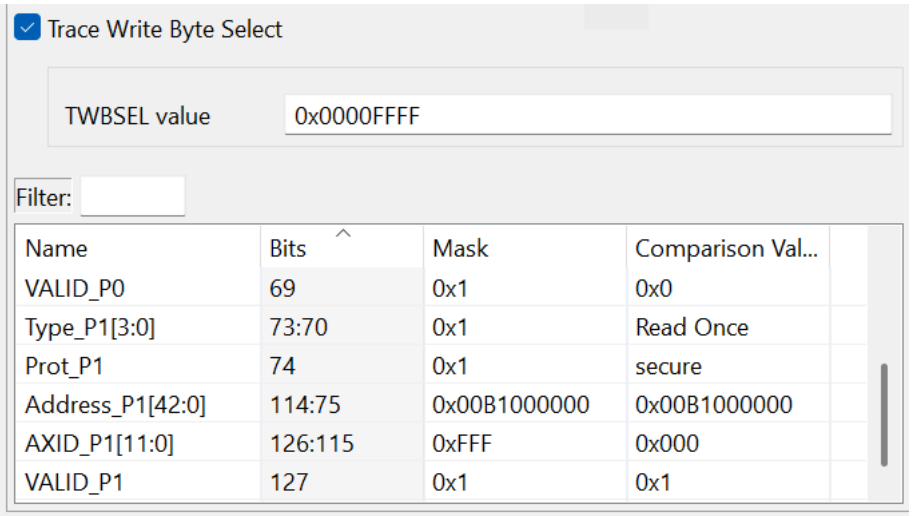


Figure 5-4: Second half of Trigger State 0 of Configure ELA



We now configure our second trigger state:

1. Open the **Trigger State 1** tab.
2. Set **Select Signal Group** value to `0x0`.

   Configuring **Select Signal Group** sets the SIGSEL1[11:0] of the Signal Select 1 register (SIGSEL1). 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. Configuring **Signal Comparison (COMP)** sets COMP[2:0] of the Trigger Control 1 register (TRIGCTRL1). 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. Configuring **Trace Capture (TRACE)** sets TRACE[7:6] of the TRIGCTRL1 register.
4. Set **Next State** to **Do not change state - final trigger state** or `0x0`.

   We set the Next State to 0, so that Trigger State 1 is our final Trigger State. Configuring **Next State** sets NEXTSTATE1 of the Next State 1 register (NEXTSTATE1).
5. In the **Action** section, tick **Enable trace**.

   This tick box enables Trigger State 1 trace capture. Configuring **Action** sets TRACE[3] of the Action 1 register (ACTION1).
6. Set **Counter compare** to `0xFFFFFFFF`.

This setting configures COUNTCOMP1[31:0] of the Counter Compare 1 register (COUNTCOMP1) to a nonzero value to ensure Trigger State 1 counts. We are using the maximum value that is 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 that is associated with the CTIIN[0] bit. Configuring **CTTRIGIN[1:0]** sets CTTRIGIN[1:0] 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. If using Arm DS 2025.0-2 or later, leave the signal mapping table to the default values.

   Leaving the default table values means we do not match on any of the signals from any of the Signal Groups.

9. For Arm DS versions earlier than 2025.0-2, 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. Configuring the **Signal Mask** and **Signal Compare** fields set SIGMASK[255:0] of the Signal Mask 1 registers (SIGMASK1) and set SIGCOMP[255:0] of the Signal Compare 1 registers (SIGCOMP1).

10. Tick **Trace Write Byte Select** and set **TWBSEL** to `0x0000FFFF`.

    This setting enables trace write for the data we are interested in. Configuring **TWBSEL** sets TRACE_BYTE[15:0] of the Trace Write Byte Select 1 register (TWBSEL1).

11. Click **Apply** and **OK**.

When the previous steps are finished, the **Trigger State 1** tab looks like the following for Arm DS 2025.0-2 and later:

Figure 5-5: First part of Trigger State 1 of Configure ELA view

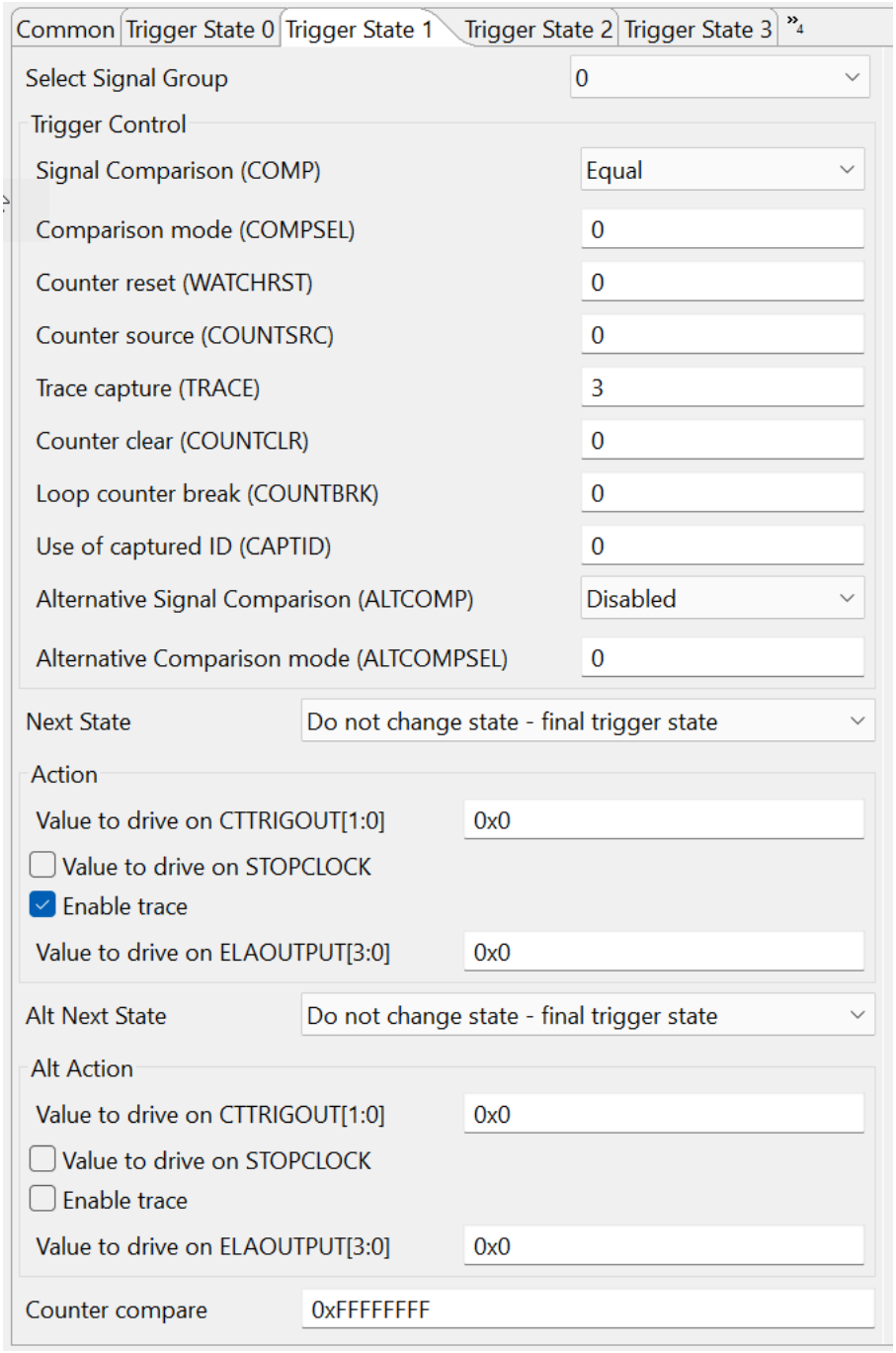


Figure 5-6: Second part of Trigger State 1 of Configure ELA view

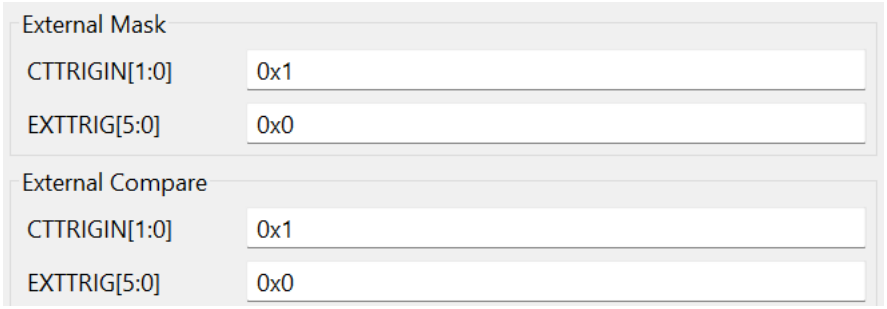
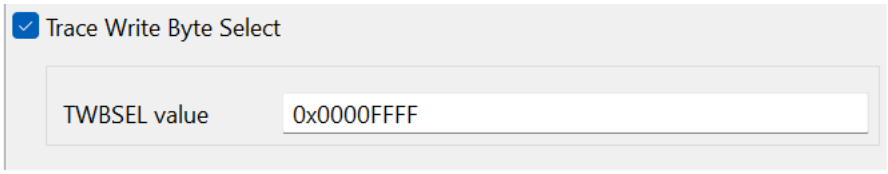


Figure 5-7: Third part of Trigger State 1 of Configure ELA view



6. Running the ELA use case scripts

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

1. To program the ELA configuration registers, navigate to **Scripts** view > **Use case** > **DTSLELA-600** > **ela_setup.py** > **Configure**.
2. Right-click **Configure ELA** and select **Run ela_setup.py::Configure ELA**.
3. 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 executing a series of Arm DS script commands. This code is available at the end of this section.
4. To run the ELA, navigate to **Scripts** view > **Use case** > **DTSLELA-600** > **ela_control.py** > **Run ELA-600**.
5. Right-click **Run ELA-600** and select **Run ela_setup.py::Run ELA-600**.

   Run ELA-600 starts the ETR by default. If your target does not use an ETR as the ELA-600 trace sink, you must:

   a. Right-click **ela_control.py::Run ELA-600** and select **Configure**.
   b. Untick **Start the connected trace sinks when the ELA-600 starts**.
   c. Start the trace sink manually or add DTSL code to the **Run ELA-600** use case script to start the trace sink.
6. Run the target with the test image.

To add the CTI associated with the ELA-600 to the Arm DS platform configuration, add the following to `dtsl_config_script.py` in `def discoverDevices(self)` after the `for coreName in (coreNames_cortexA55)` loop:

```
cti = CSCTI(self, self.findDevice("CSCTI_3"), "CSCTI_3")  ##Existing CTI
 instantiation

#Enable input/output events

cti.enableInputEvent( 1, 0)   #Where channel is the CTM channel to use for triggered
 events

cti.enableOutputEvent( 1, 0)

cti = self.getDeviceInterface("CSCTI_3")  #Where "CSCTI_1" is the name of the CTI
 device in the .sdf/rcf/rcv file
```

To setup and enable the CTI connection between the core on the system and the ELA-600, execute the following as an Arm DS script:

```
echo ***** Set Level output *****
```

```
echo memory set_typed APB_0:0x80920014 (unsigned int) (0x00000001)

echo ***** Setting up CTI4 CORE SIDE *****
echo ***** Set CTI4 CTIOUTEN0 -> Map channels in CT system to trigger outputs*****
echo Set CTIOUTEN (CTI Channel to Trigger 0 Enable register) @ DP_1 AP_0
 (APB):0x820200A0 => 0x000000F
 memory set_typed APB_0:0x820200A0 (unsigned int) (0x0000000F)

echo ***** Set CTI4 CTIINEN0 -> Map Trigger Inputs to Channels in CT System *****
echo Set CTIINEN0 (CTI Trigger 0 to Channel Enable register) @ DP_1 AP_0
 (APB):0x82020020 => 0x000000F
memory set_typed APB_0:0x82020020 (unsigned int) (0x0000000F)

echo ***** Enabling CTI4 CTRL Register ****
echo Set CTICTRL (CTI CTRL register) @ DP_1 AP_0 (APB):0x82020000 => 0x0000001
memory set_typed APB_0:0x82020000 (unsigned int) (0x00000001)

echo ***** Setting up CT3 ELA SIDE ****
echo ***** Enabling CTI3 CTRL Register ****
echo Set CTICTRL (CTI CTRL register) @ DP_1 AP_0 (APB):0x80920000 => 0x0000001
memory set_typed APB_0:0x80920000 (unsigned int) (0x00000001)

echo ***** Enabling CTI3 CTIINEN0 -> Map Trigger Inputs to Channels in CT
 System  ****
echo Set CTIINEN0 (CTI Trigger 0 to Channel Enable register) @ DP_1 AP_0
 (APB):0x80920020 => 0x000000F
memory set_typed APB_0:0x80920020 (unsigned int) (0x000000F)

echo ***** Set CTI3 CTIOUTEN0 -> Map channels in CT system to trigger outputs *****
echo Set CTIOUTEN (CTI Channel to Trigger 0 Enable register) @ DP_1 AP_0
 (APB):0x820200A0 => 0x000000F
memory set_typed APB_0:0x809200A0 (unsigned int) (0x0000000F)
```

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.

7. Capturing the ELA trace data

The following instructions show 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 corrupt the memory contents at address `0xB1000000`, execute the following as an Arm DS script:

   ```
   echo ***** Corrupting 0xB1000000 & xB1000004 *****
   memory set AXI_0:0xB1000000 0 0xFFFFFFFF
   memory set AXI_0:0xB1000004 0 0xFFFFFFFF
   ```

2. To stop the ELA, navigate to **Scripts** view > **Use case** > **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 target does not use an ETR as the ELA-600 trace sink, you must:

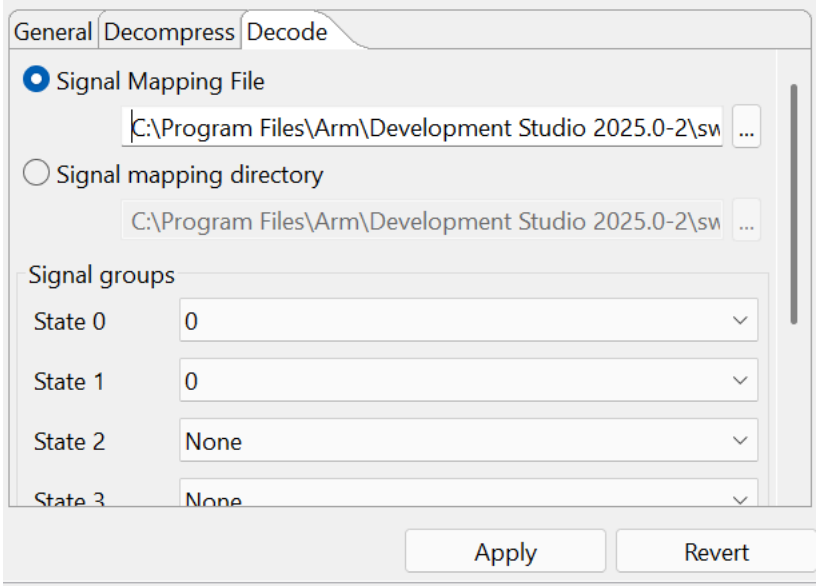
   a. Right-click **ela_control.py::Stop ELA-600** and select **Configure**.
   b. Untick **Stop the connected trace sinks when the ELA-600 stops**.
   c. Stop the trace sink manually or add DTSL code to the **Stop ELA-600** use case script to stop the trace sink.
4. To dump and decode the ELA trace, navigate to **Scripts** view > **Use case** > **DTSLELA-600** > **ela_process_trace.py** > **Decompress and decode ELA trace**.
5. Right-click on **Decompress and decode ELA trace** and select **Configure**.
6. Under the **General** tab, select **Decompress and decode trace**.
7. 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.
8. In the **Decode** tab:

   a. If using Arm DS 2025.0-2 or later, select **Signal Mapping File** and set the file path to your signal mapping JSON file. For this example, leave the default JSON file path to *axi_interconnect_mapping.json*.
   b. For Arm DS versions earlier than 2025.0-2, set **ELA trace mapping file** to **axi_interconnect_mapping.json** in the **DTSLELA-600** project.

      Note: If you are using your own JSON file, enter the JSON file path and name.
   c. 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 7-1: Decode tab settings



9. Click **Apply** and **OK**.
10. 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.

8. Analyzing the ELA trace data

As a result of the previous steps, the ELA traces 3 transactions to address `0xB1000000` and some counter values. In this example, the trace data is output to the ETR and the following 3 transactions to address `0xB1000000` are generated:

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 acknowledgment signal.

The following generated trace capture shows the decompressed and decoded trace data for the previously mentioned 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:
 0x0000000300000003000000030000000B
```

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

9. Resources

The following resources are related to this guide:

- Application Note - Arm CoreSight ELA-600 Version 1.0
- CoreSight ELA-600 Product Support
- Arm CoreSight ELA-600 Embedded Logic Analyzer Technical Reference Manual (TRM)
- “Embedded Logic Analyzer” section of the Arm Development Studio User Guide
- How do I program the conditions for the ELA-600 “trigger state” state machine to move to the next state?