Arm Debugger Manual Configuration

Version 1.0

Non-Confidential

Issue 01

102551_0100_01_en

Arm Debugger Manual Configuration

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	21 July 2021	Non-Confidential	First release

Proprietary Notice

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.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

The aim of this workbook is to manually create a platform configuration for a given target with Arm Development Studio's Platform Configuration Editor (PCE).

For the majority of targets, you can create a platform configuration automatically by performing target auto-detection with PCE. This means that manually configuring a target from start to finish is rarely required.

However, manually configuring a target can help you understand:

- The information required to create a platform configuration.
- How a platform configuration is created.
- Which CoreSight devices are associated with debug and trace.
- How and why CoreSight devices are connected together.
- Important settings for the CoreSight devices.

To complete this tutorial, you require:

- An Arm Development Studio installation of version 2019.0 or later.
- Have a basic understanding of Arm system debug and trace.

2. Understanding CoreSight

CoreSight technology is the Arm solution for debug and trace in complex SoC designs. CoreSight consists of:

- A library of modular devices and component interconnects.
- Architected discovery and identification methods to allow for flexible system design.
- A standard for implementing the Arm Debug Interface for debug tools.

CoreSight provides the ability to read and modify register values of CPUs and peripherals and provides monitoring and triggering resources.

CoreSight trace allows for the continuous collection of system information for later analysis and includes:

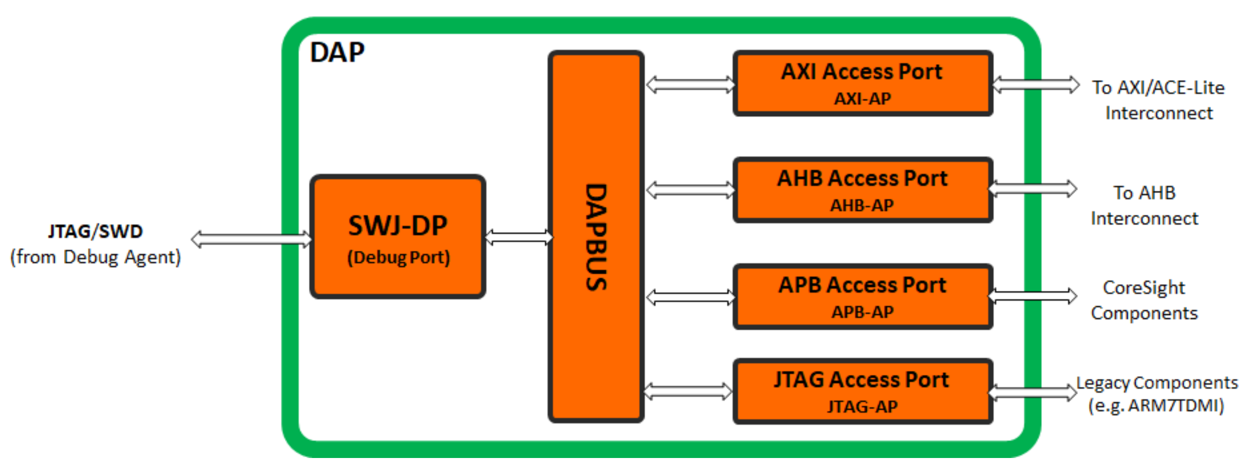
- trace sources such as the Embedded Trace Macrocell (ETM).
- trace links such as the funnel and the replicator.
- trace sinks such as the Trace Memory Controller (TMC) Embedded Trace FIFO (ETF), the TMC Embedded Trace Router (ETR), and the Trace Port Interface Unit (TPIU).

Typically, CoreSight devices are behind a CoreSight Debug Access Port (DAP). A DAP presents a physical port to be connected to by external debug tools either using JTAG or Serial Wire Debug (SWD). A DAP is a DP connected to one or more Access Ports (APs or MEMAPs). The MEMAP types available are:

- Advanced Peripheral Bus Access Port (APB-AP).
- Advanced High Performance Bus Access Port (AHB-AP).
- Advanced eXtensible Interface Access Port (AXI-AP).

In the following you can see a diagram of a DAP:

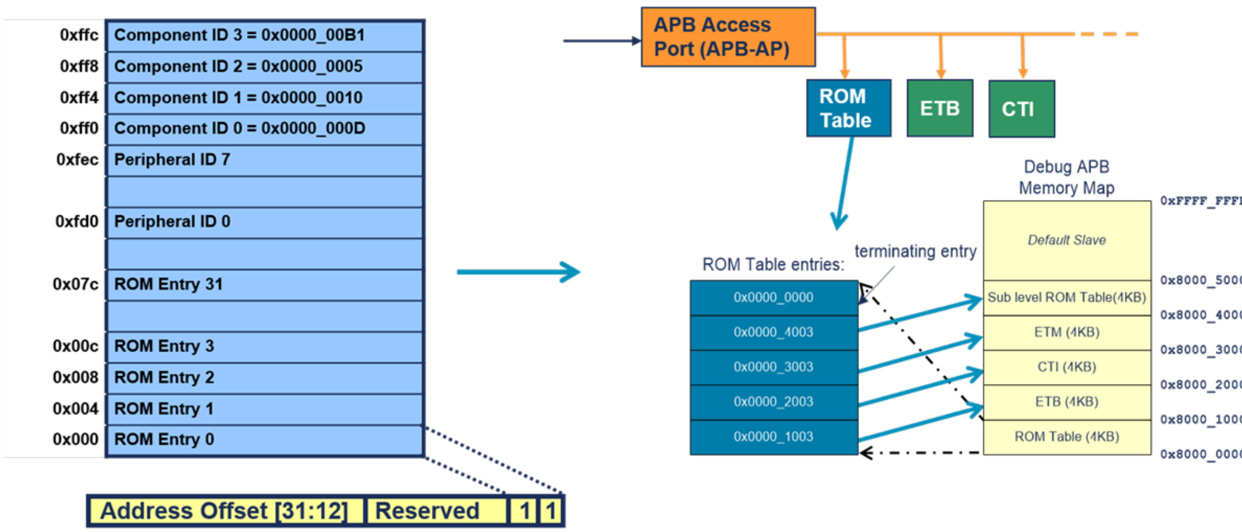
Figure 2-1: Diagram of a DAP



All CoreSight systems include at least one ROM table. The ROM table allows an external debugger to discover the CoreSight devices on the target. Each entry in the ROM table contains an address offset that points to the base address of a device accessible through the MEMAP or another ROM table.

In the following you can see a diagram of a ROM table:

Figure 2-2: ROM Table



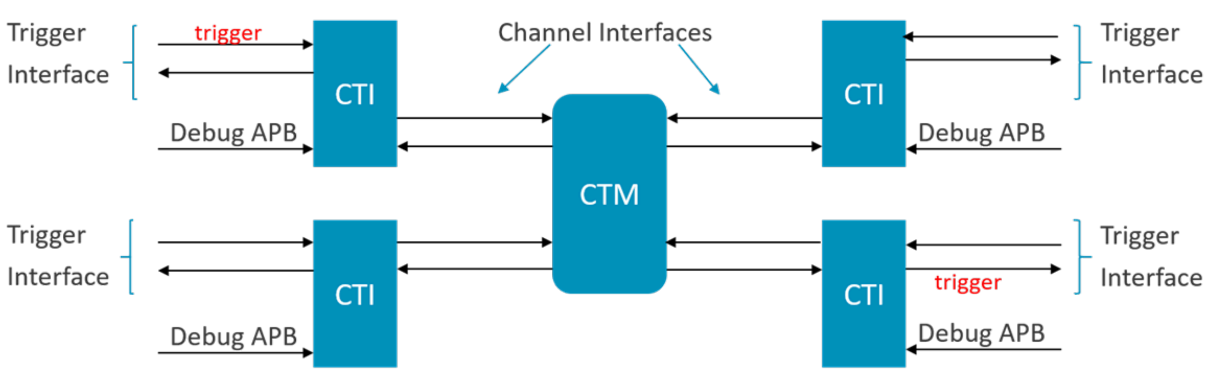
Arm systems have an Embedded Cross Trigger (ECT) that consists of Cross Trigger Interfaces (CTIs) and Cross Trigger Matrixes (CTMs).

CTIs send and receive trigger events through the Trigger Interface. Trigger events are mapped to channel events and transmitted through the Channel Interface. CTIs have programmable mappings between triggers and channels.

CTMs broadcast channel events through Channel Interfaces and enable the linking of CTIs.

In the following you can see a diagram of an ECT:

Figure 2-3: Diagram of an ECT



Typically, the ECT is used for:

- Cross-halting CPUs.
- Simultaneous CPU restart.
- Trace collection trigger.
- Interrupt generation.
- Cross component mapping between CPU and FPGA subsystems.

3. Understanding a target's debug and trace infrastructure

In the Understanding CoreSight section, you learned that CoreSight devices are used to enable debug and trace capability for a target. In this section, we are going to look at what target information is required to create a platform configuration in Arm Development Studio.

In order to manually create a platform configuration for a target, you must know:

- All the scan chain and CoreSight devices present.
- The type and number of DPs.
- The type, number, and index values of all APs.
- How the different devices are connected together (known as the CoreSight topology).
- Device-specific information such as implementation settings.

This information is usually found in tabular or block diagram form in the target's documentation.

This tutorial focuses on an example target's debug and trace infrastructure. The target contains a two-core Cortex-A72 cluster, a four-core Cortex-A53 cluster, and a Cortex-M3. The target is modeled on an Arm Development Platform and some CoreSight devices are left out intentionally for the purpose of this tutorial. The target conforms to the Arm Debug Interface Architecture Specification ADIv5.0 to ADIv5.2 implemented by the Arm CoreSight SoC-400 Technical Reference Manual. CoreSight SoC-400 uses AP version 1 (APv1).

The table below lists the debug and trace infrastructure devices and component connections for the target described above:

Note

S: = Slave

<number>: = Trigger or Slave value

Device Type	PCE device name	AP index	CoreSight Base Address	Connected to
DP	ARMCS-DP_0	NA	NA	
AXI-AP	CSMEMAP (0: AXI-AP) - APv1	0	NA	ARMCS-DP_0
APB-AP	CSMEMAP (1: APB-AP) - APv1	1	NA	ARMCS-DP_0
TMC (ETF)	CSTMC_0	1	`0x80010000`	S:0: CSTFunnel_2
CTI	CSCTI_7	1	`0x80020000`	S:0: CSTMC_0 S:1: CSTMC_1 S:3: CSTPIU
TPIU	CSTPIU	1	`0x80030000`	
Funnel	CSTFunnel_1	1	`0x80040000`	S: CSTMC_0

Device Type	PCE device name	AP index	CoreSight Base Address	Connected to
TMC (ETR)	CSTMC_2	1	0x80070000	
Replicator	CSATBReplicator	1	0x80120000	S:0: CSTPIU S:1: CSTMC_1
Funnel	CSTFunnel_2	1	0x80150000	S: CSATBReplicator
Cortex-A72	Cortex-A72_0	1	0x82010000	S:1: CSCTI_0
CTI	CSCTI_0	1	0x82020000	
Cortex-A72	Cortex-A72_1	1	0x82110000	S:1: CSCTI_1
CTI	CSCTI_1	1	0x82120000	
Cortex-A53	Cortex-A53_0	1	0x83010000	S:1: CSCTI_2 S: CSETM_0
CTI	CSCTI_2	1	0x83020000	
ETM	CSETM_0	1	0x83040000	S:0: CSTFunnel_0
Cortex-A53	Cortex-A53_1	1	0x83110000	S:1: CSCTI_3 S: CSETM_1
CTI	CSCTI_3	1	0x83120000	
ETM	CSETM_1	1	0x83140000	S:1: CSTFunnel_0
Cortex-A53	Cortex-A53_2	1	0x83210000	S:1: CSCTI_4 S: CSETM_2
CTI	CSCTI_4	1	0x83220000	
ETM	CSETM_2	1	0x83240000	S:2: CSTFunnel_0
Cortex-A53	Cortex-A53_3	1	0x83310000	S:1: CSCTI_5 S: CSETM_3
CTI	CSCTI_5	1	0x83320000	
ETM	CSETM_3	1	0x83340000	S:3: CSTFunnel_0
Funnel	CSTFunnel_0	1	0x830C0000	S:0: CSTFunnel_1
AHB-AP-M	CSMEMAP (2: AHB-AP-M) - APv1	2	NA	ARMCS-DP_0
Cortex-M3	Cortex-M3	2	NA	S:7: CSCTI_6
CTI	CSCTI_6	2	0xE0044000	

4. Set up the platform configuration manually

In this section, we show how to open PCE in the Arm Development Studio IDE. At the end of this section, you will be ready to manually create a platform configuration.

Platform configurations are stored in a configuration database (configDB). We enter PCE by right-clicking on a configDB displayed in the Project Explorer view.

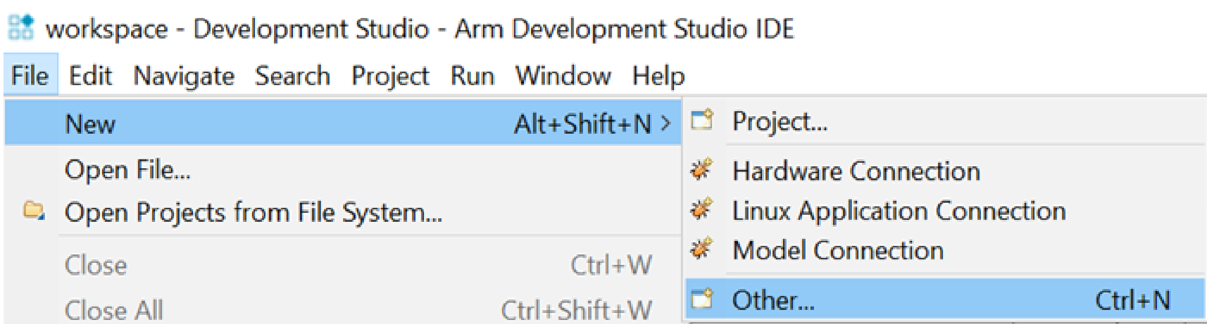
Go to Create a platform configuration section if you have a configDB you have created or a configDB called ExtensionDB listed in the Project Explorer view.

Continue with the Create a configDB section if you do not have a configDB to work with.

Create a configDB

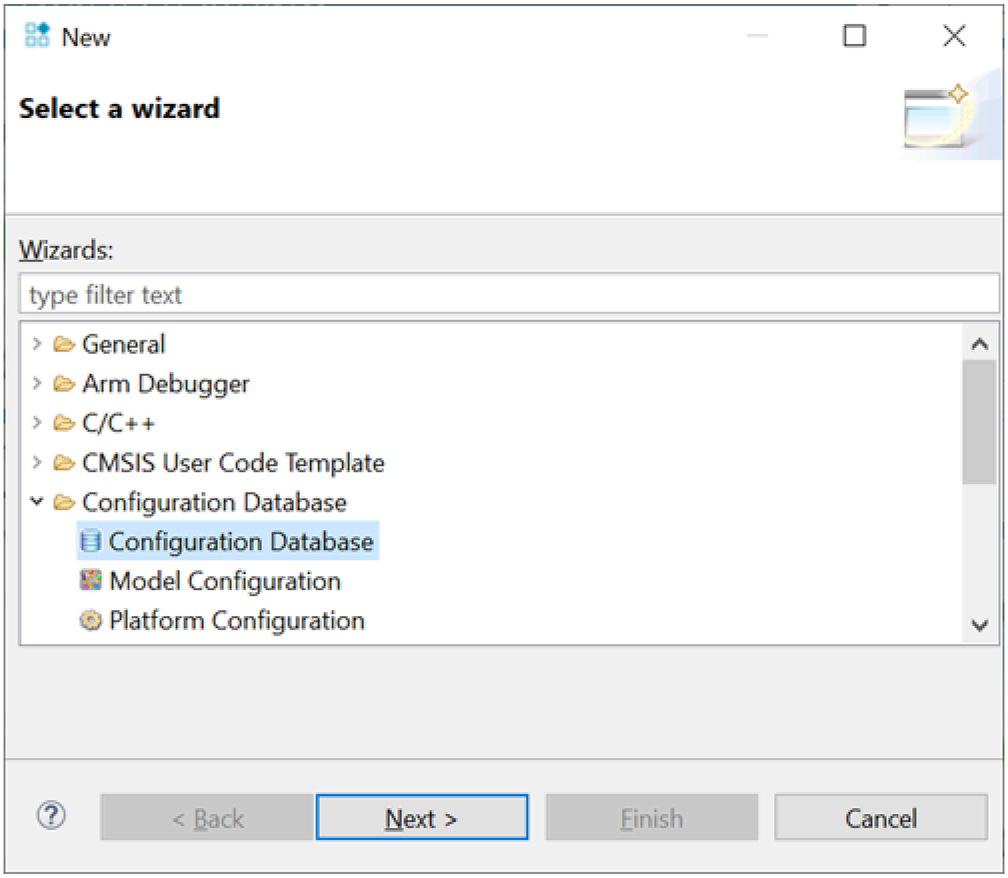
1. Launch Arm Development Studio IDE and select **File** > **New** > **Other**....

   Figure 4-1: Using File menu to add a configDB

2. Select **Configuration Database** > **Configuration Database**.

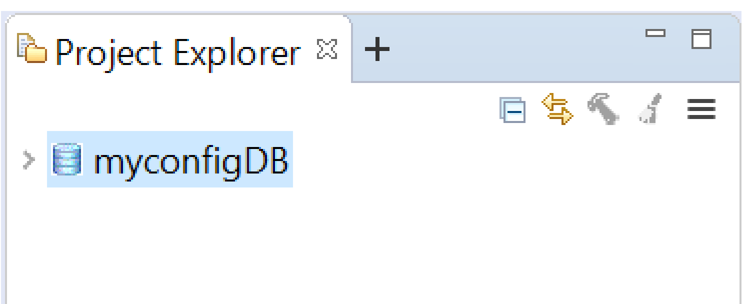
Figure 4-2: Creating a new configDB



3. Click **Next**.
4. Enter a **Database Name** and click **Finish**.

   In the Project Explorer view, a configDB with the name you choose appears.

Figure 4-3: A created configDB in the Project Explorer view

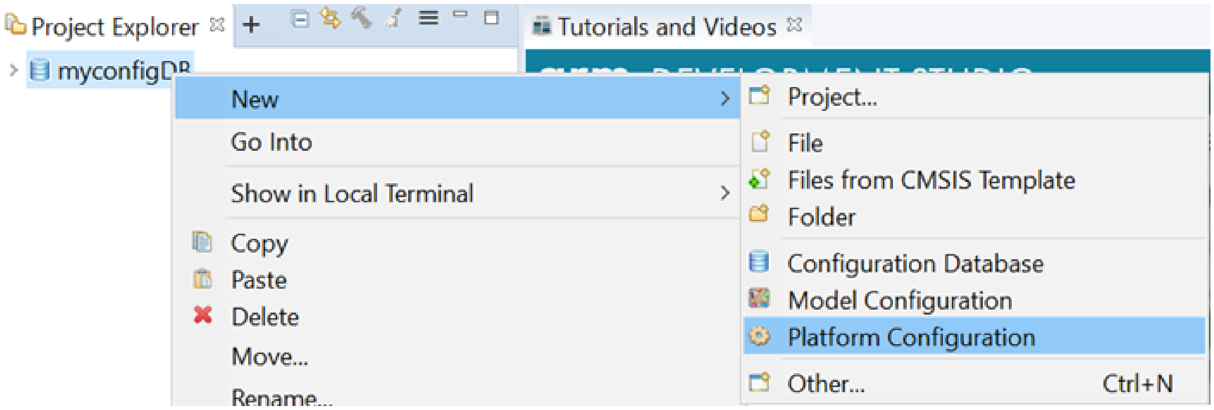


Create a platform configuration

Using a configDB, create a platform configuration.

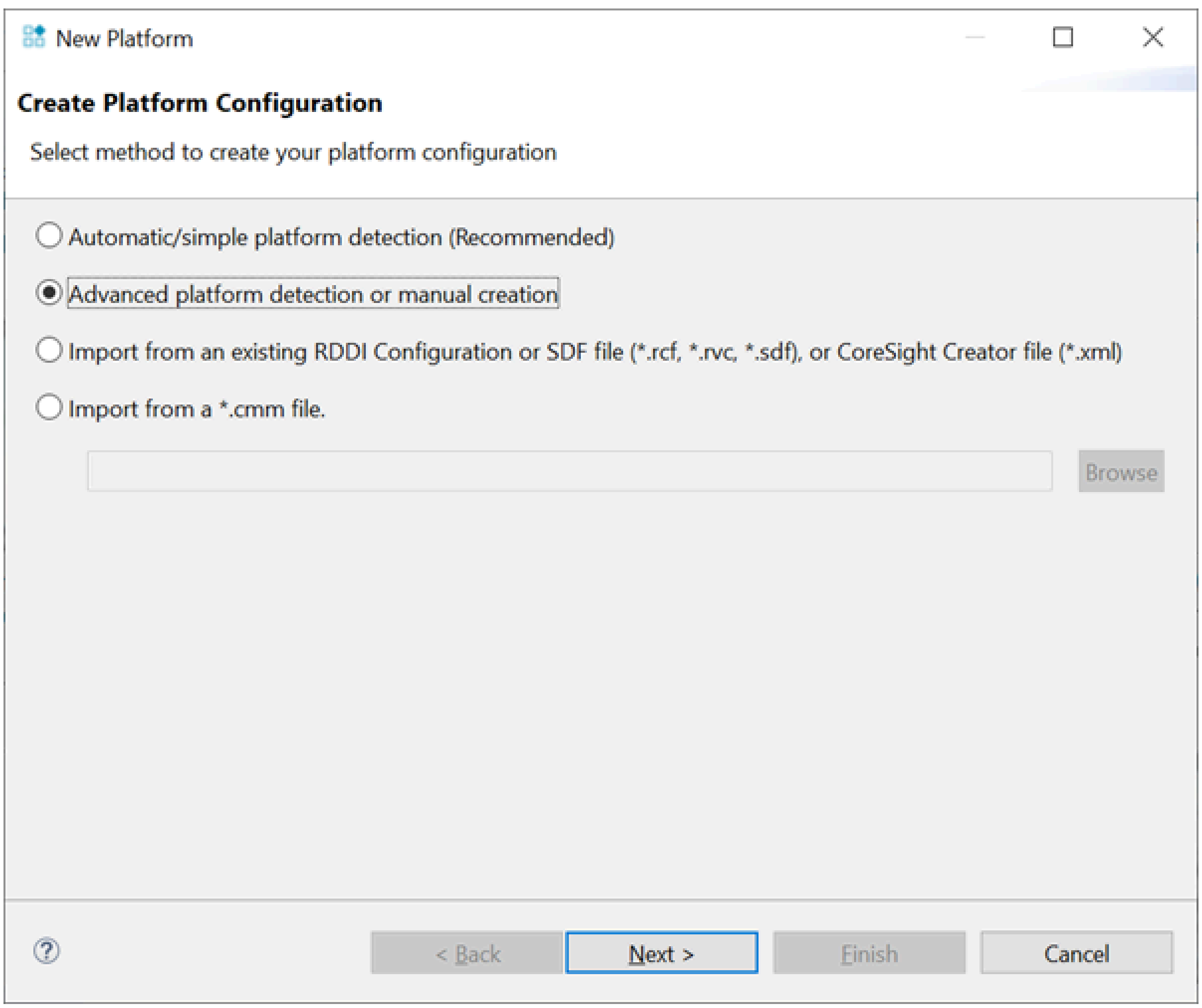
1. In the **Project Explorer**, right-click on the **configDB** and select **New** > **Platform Configuration**.

Figure 4-4: Creating a new platform configuration



2. Select **Advanced platform detection** or **manual creation**.

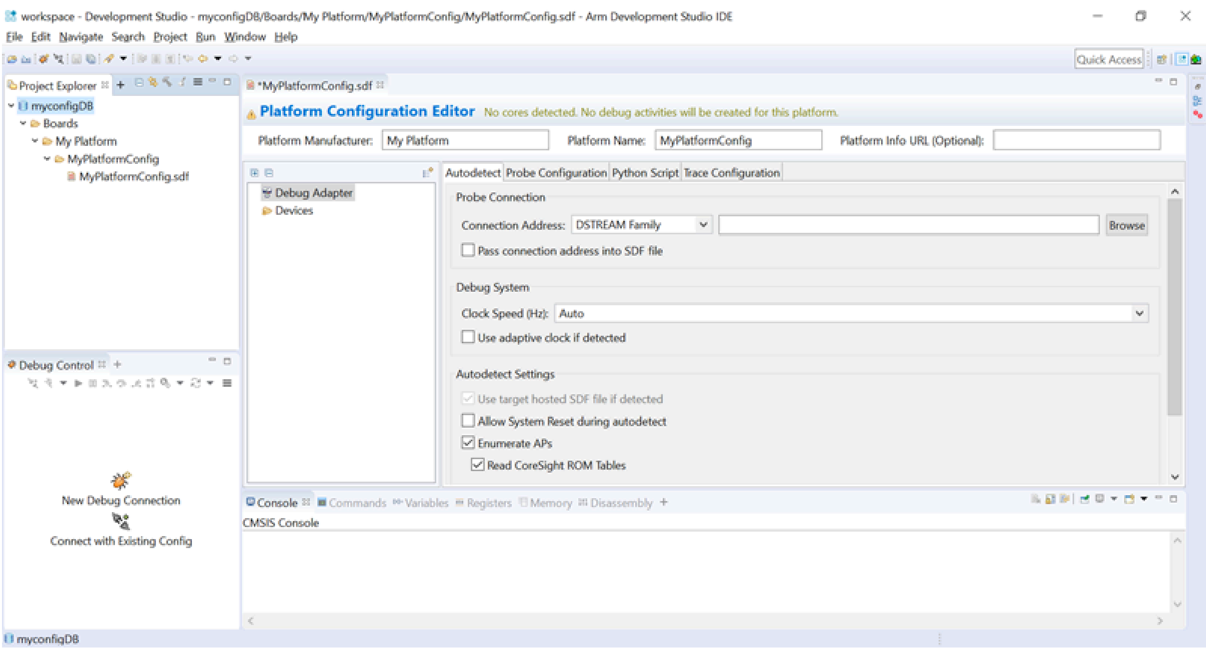
Figure 4-5: Create a platform configuration using manually configuration



3. Click **Next**.
4. Enter the platform information and click **Finish**.

   A system description file (SDF) opens. The SDF is set up with the platform information you provided. To view the hierarchy of the platform configuration, in the Project Explorer view, expand the configDB.

Figure 4-6: A created platform configuration SDF file



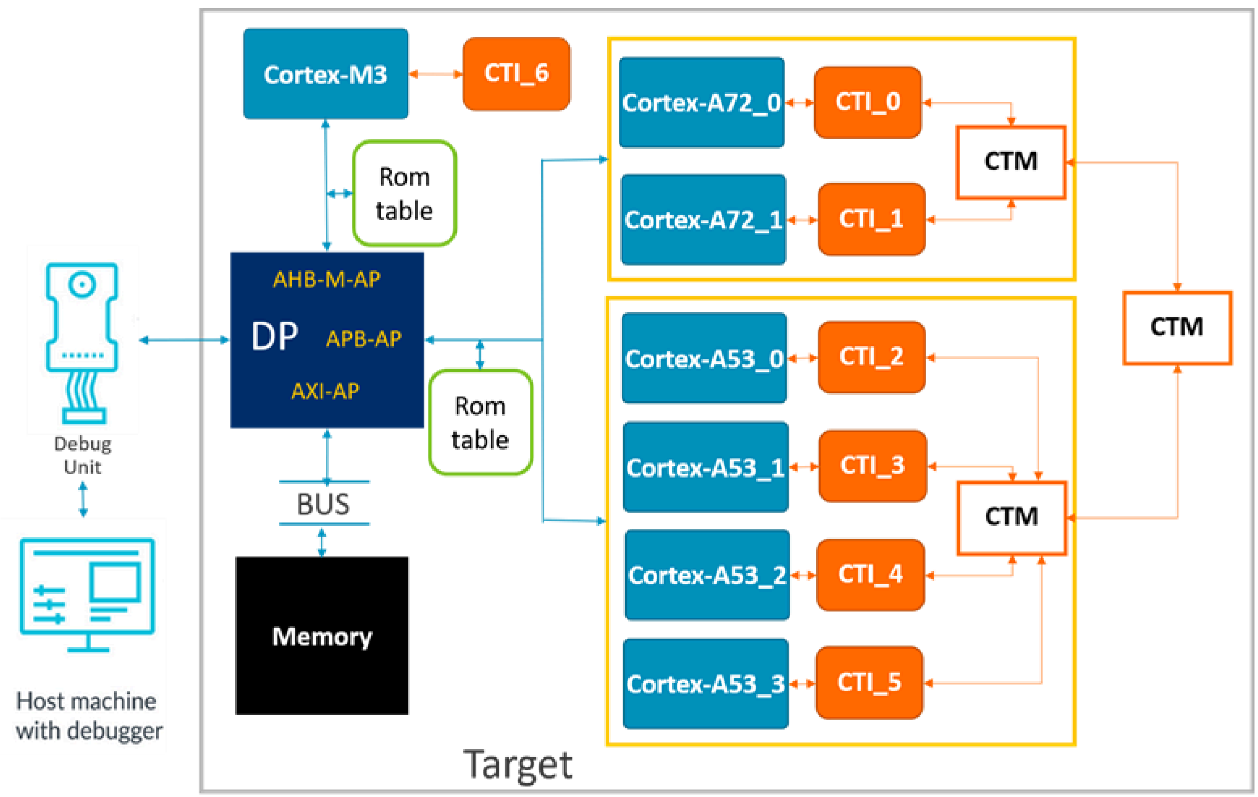
We are now ready to add our target's information to our new platform configuration.

5. Manually configuring a platform configuration for debug

In this section, we manually create a platform configuration for the target described in the Set up the platform configuration manually section. We focus on creating a platform configuration we can debug only first, so that we know we have a working configuration before we move on to adding trace capability in the next section.

Below is a block diagram of the debug-related devices for the board we are manually configuring:

Figure 5-1: Diagram of the example target's debug infrastructure



The debug-specific details of this target are:

- All the target's CoreSight devices are behind a DP.
- The DP has three APs:
    - An AXI-AP which enables direct access to the board's system memory via the DP.
    - An APB-AP which grants access to the CoreSight components for the Cortex-A72 and Cortex-A53 clusters.This AP has a ROM table.
    - An AHB-M-AP which grants access to the Cortex-M3 and its associated CoreSight devices.This AP has a ROM table.

- A Cortex-A72 cluster containing 2 Cortex-A72 cores.
- A Cortex-A53 cluster containing 4 Cortex-A53 cores.
- Each core has an associated CTI.

Each cluster has a Cross Trigger Matrix (CTM) to connect that cluster's CTIs together.

A CTM to connect the cluster CTMs together to enable cross-cluster synchronization.

It is important to note that the Cortex-M3 associated CTI is not connected to either CTM, so synchronizing debug operations between the clusters and the Cortex-M3 is not possible.

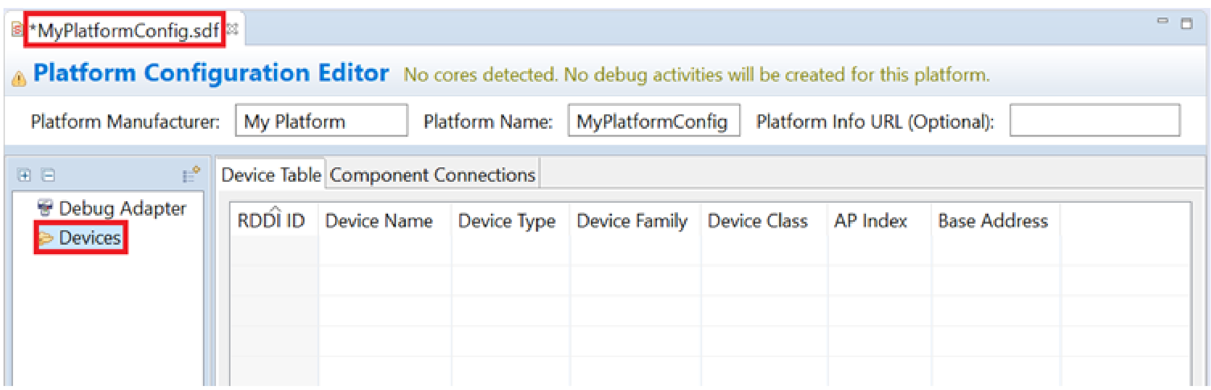
We now start adding devices to the platform configuration.

Add a DP and APs to the platform configuration

In this section, we add a DP, an AXI-AP, and an APB-AP to the platform configuration.

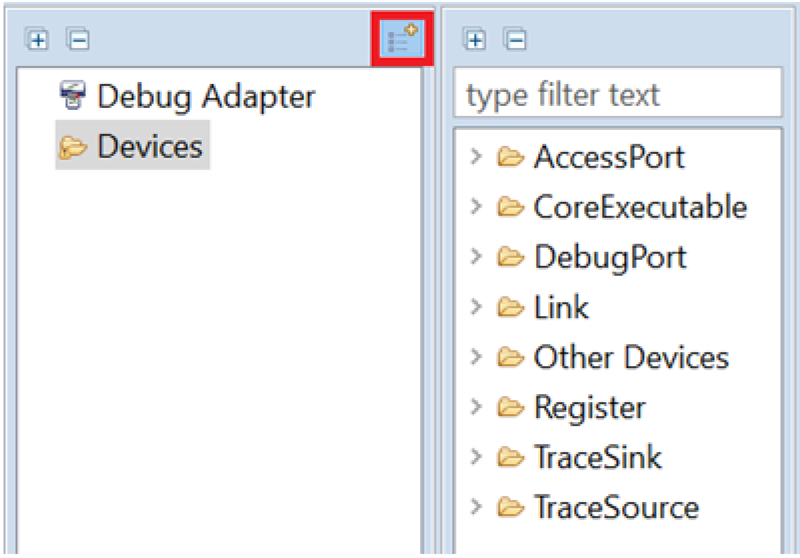
1. Click on **Devices** in the SDF file.

Figure 5-2: A SDF file with an empty Devices list



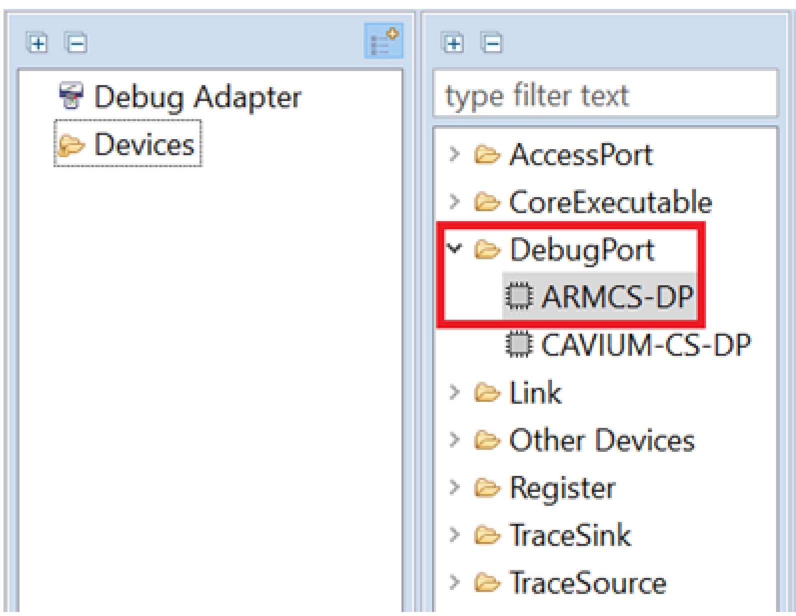
2. Click **Toggle Devices Panel**.

Figure 5-3: Toggling the Devices Panel



3. Click **DebugPort** > **ARMCS-DP**.

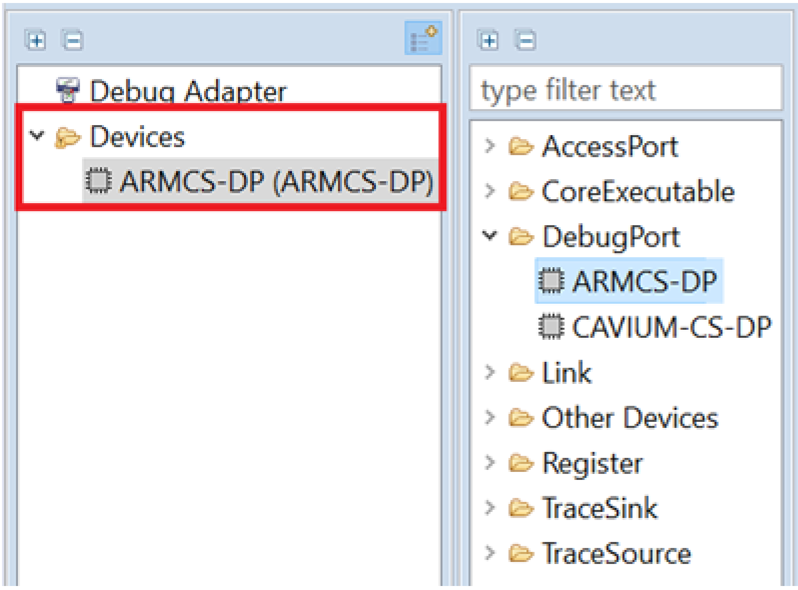
Figure 5-4: Selecting an ARMCS-DP



4. Drag **ARMCS-DP** to **Devices**.

   This adds a **ARMCS-DP** device to the **Devices** list:

Figure 5-5: Adding an ARMCS-DP

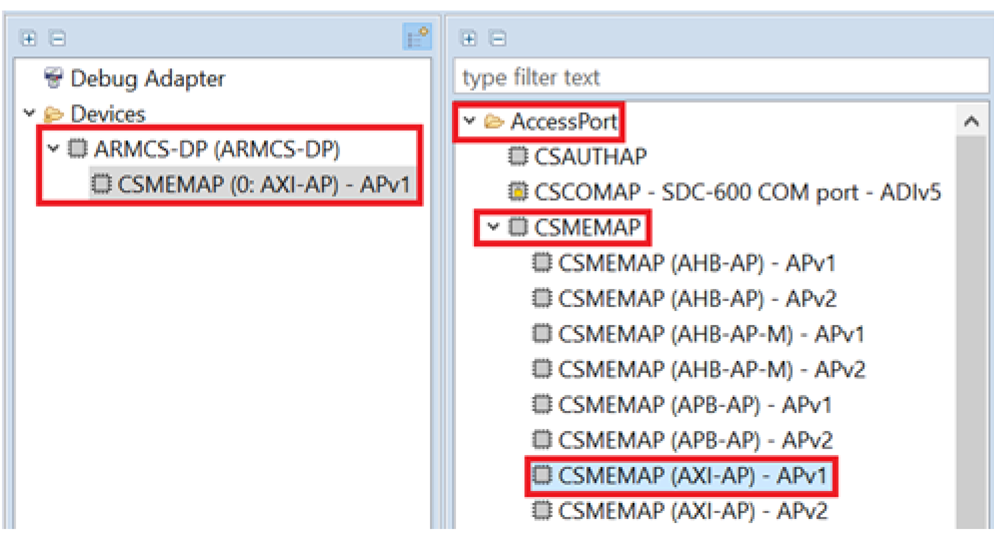


Repeat a similar process for every device added.

5. Add an **AXI-AP** to the **ARMCS-DP**.

   Add a **CSMEMAP (0: AXI-AP) - APv1** to the **ARMCS-DP**:

Figure 5-6: Adding a CSMEMAP (AXI-AP) - AP1



6. Click on CSMEMAP (0: AXI-AP) under ARMCS-DP.

   There are certain settings which must be correct for Arm Debugger to connect and debug the board.

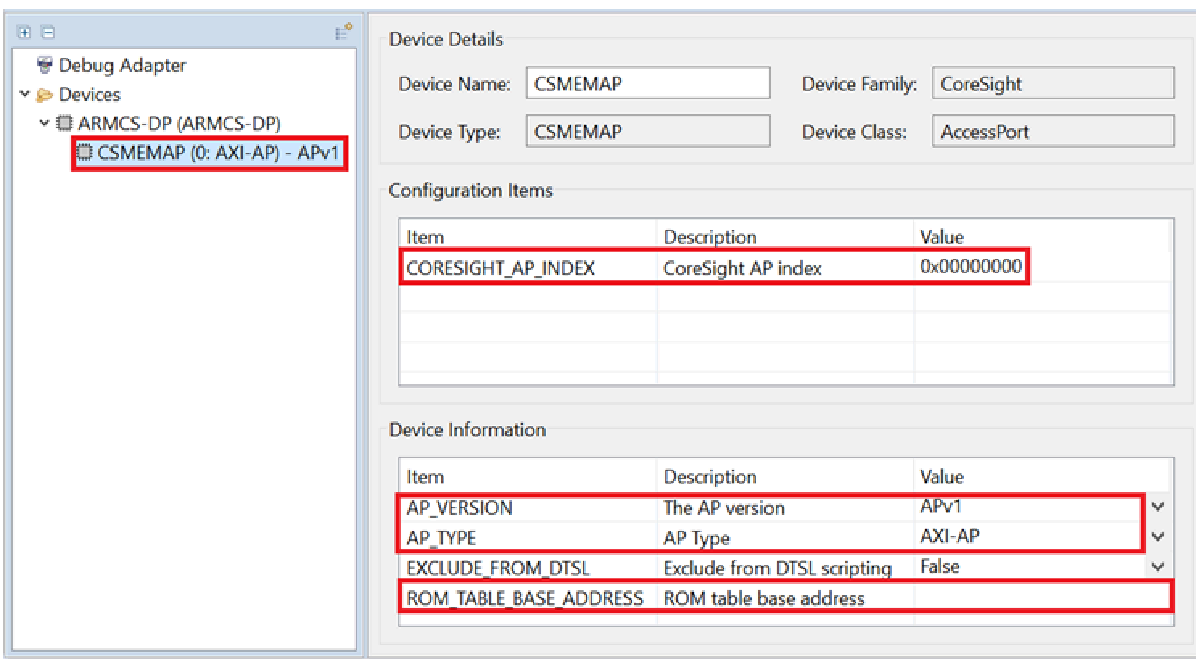
   - **CORESIGHT_AP_INDEX**

    - The AP's index number on the DP.
- **AP_VERSION**
    - The version of the architecture the AP implements.
- **AP_TYPE**
    - The type of the AP for the MEM-AP.
- Optional, **ROM_TABLE_BASE_ADDRESS**
    - The base address of the ROM table for the AP.
    - This is optional as setting the ROM table is not necessary for manual configuration.You want to set this if:
        - The ROM table base address reported by the target is incorrect.
        - You are going to auto-detect the devices attached to an AP after manually adding the AP.

For our board, the details for AP0 are:

- **CORESIGHT_AP_INDEX** is `0x0`.
- **AP_VERSION** is **APv1]{.ui}.
- **AP_TYPE** is **AXI-AP]{.ui}.
- **ROM_TABLE_BASE_ADDRESS** is left empty as there is no ROM table on this AP.

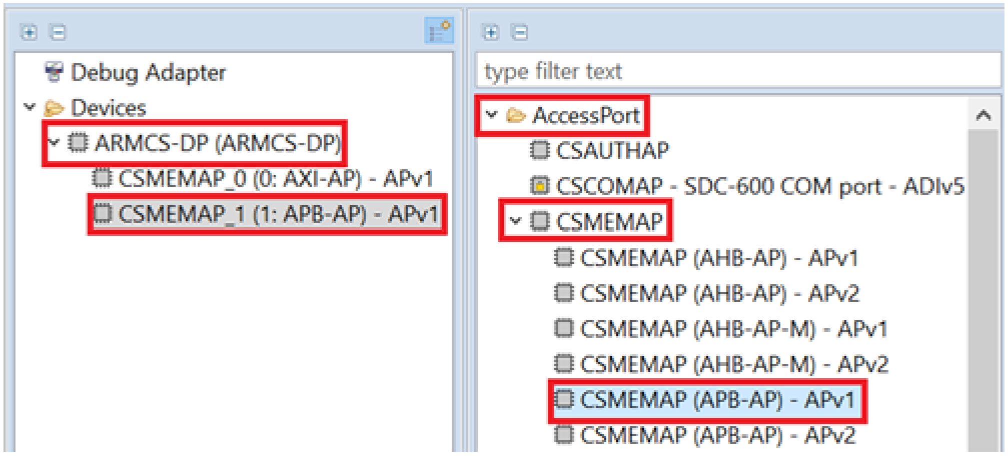
Figure 5-7: Setting up the CSMEMAP (AXI-AP) - AP1



7. Add an APB-AP to the **ARMCS-DP**.

   Add a CSMEMAP (1: APB-AP) - APv1 to the ARMCS-DP:

Figure 5-8: Adding a CSMEMAP (APB-AP) - AP1



8. Click on **CSMEMAP (1:APB-AP)** under **ARMCS-DP**.

   For our board, the details for AP1 are:

   - **CORESIGHT_AP_INDEX** is `0x1`.
   - **AP_VERSION** is **APv1**.
   - **AP_TYPE** is **APB-AP**.
   - **ROM_TABLE_BASE_ADDRESS** is `0x80000000`.

Note on enumerating APs

After adding a DP to the platform configuration, you can choose to use the PCE auto-detection process to enumerate the DP's APs rather than adding the APs manually.

To enumerate the APs:

1. In the SDF under **Debug Adaptor** > **Autodetect** > **Probe Connection**, set the **Connection Address** to the correct debug unit type and browse for the TCP or USB address of the debug unit connected to the target.
2. **Under Devices**, right-click on the **DP**.
3. Select **Enumerate APs**.

All the found APs appear under the DP.

In this tutorial, the APs are added manually.

Note on reading ROM table(s)

After adding an AP to the platform configuration, you can choose to use the PCE auto-detection process to read the AP's ROM table(s). Reading the ROM table(s) has PCE read each ROM table entry, determine the devices listed, and add the determined devices to the platform configuration. This process replaces having to manually add the AP devices.

To read the AP's ROM table(s):

1. In the SDF under **Debug Adaptor** > **Autodetect** > **Probe Connection**, set the **Connection Address** to the correct debug unit type and browse for the TCP or USB address of the debug unit connected to the target.
2. **Under Devices**, right-click on an **AP**.
3. Select **Read CoreSight ROM Tables**.

All the determined CoreSight devices appear under the AP.

In this tutorial, the AP devices are added manually.

Add debug devices and component connections to the platform configuration

In this section, we add the below to the platform configuration:

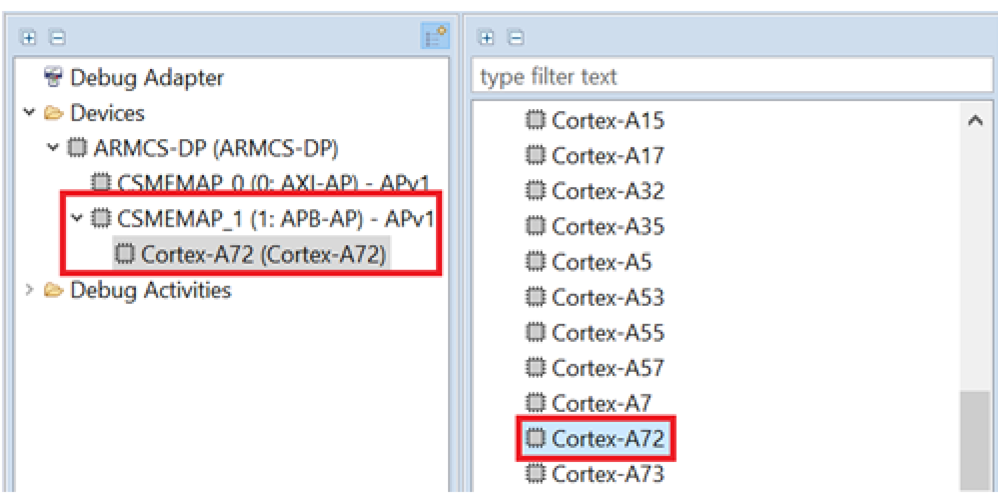
1. A Cortex-A72 core.
2. A CTI for the Cortex-A72 core.
3. A component connection between the Cortex-A72 core and its CTI.
4. The rest of the debug-related devices and component connections.

This section also covers the generated platform configuration debug activities such as bare-metal and SMP debug connections. Additionally, this section lists good tests to check whether the platform configuration allows you to successfully debug a target with the Arm Debugger.

1. In the Devices Panel, select CoreExecutable > Cortex-A72 and drag it to CSMEM-AP_1.

   Add a Cortex-A72 to CSMEMAP_1:

   Figure 5-9: Adding a Cortex-A72

2. Click on Cortex-A72 under CSMEMAP_1.

   There are certain core settings which must be correct for Arm Debugger to connect and debug the core.

   - **CORESIGHT_BASE_ADDRESS**

- The lower 32-bits of the CoreSight base address for the core.
- **CORESIGHT_BASE_ADDRESS_MSW**
  - The higher 32-bits of the CoreSight base address for the core.
- **CTI_CORESIGHT_BASE_ADDRESS**
  - The CoreSight base address for the CTI associated with the core.
- **CTI_SYNCH_START**
  - Whether the CTI can be used for synchronizing execution.

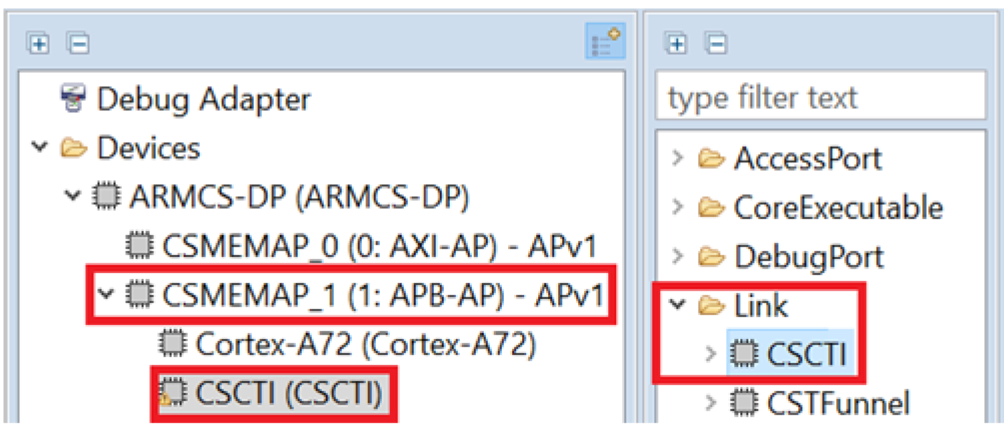
For our board, the details for the first Cortex-A72 are:

- **CORESIGHT_BASE_ADDRESS** is `0x82010000`.
- **CORESIGHT_BASE_ADDRESS_MSW** is `0x0`.
- **CTI_CORESIGHT_BASE_ADDRESS** is `0x82020000`.
- **CTI_SYNCH_START** is **True** as the core has an associated CTI.

3. Add a CTI for Cortex-A72.

Add a CTI to the CSMEMAP_1:

Figure 5-10: Adding a CSCTI



4. Click on **CTI** under **Cortex-A72**.

There are certain CTI settings which must be correct for Arm Debugger to make use of the CTI for synchronous execution.

- **CORESIGHT_BASE_ADDRESS**
  - The lower 32-bits of the CoreSight base address for the CTI.
- **CORESIGHT_BASE_ADDRESS_MSW**
  - The higher 32-bits of the CoreSight base address for the CTI.
- **SYNCH_START_ENABLE**
  - Enables synchronized execution.For example, use the CTI to start and stop the associated core.

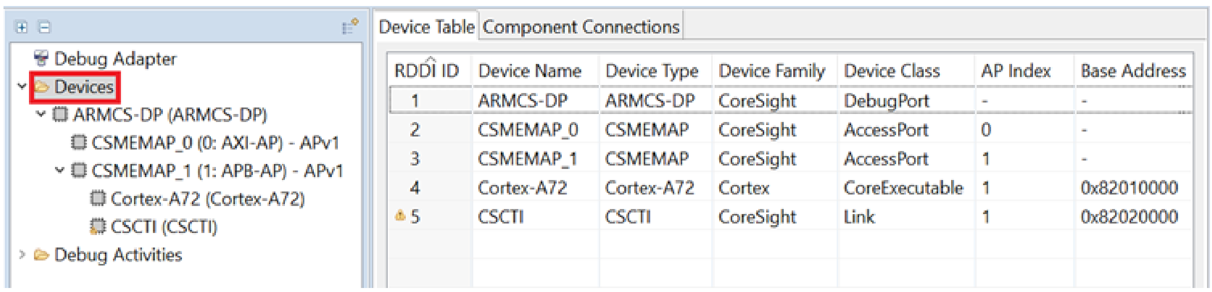
- **SYNCH_START_CHANNEL**
    - Which CTI channel is associated with starting core execution.

For our board, the details for the first CTI are:

- **CORESIGHT_BASE_ADDRESS** is `0x82020000`.
- **CORESIGHT_BASE_ADDRESS**_MSW is `0x0`.
- **SYNCH_START_ENABLE** is **True** as CTI is used for synchronous core starting.
- **SYNCH_START_CHANNEL** is `1` as synchronous starting is linked to CTI channel 1.

5. Click **Devices** to view the details of the devices you have added.

Figure 5-11: List of current devices in the Devices tab



To debug a target, Arm Debugger must know how the components are connected. We add this connection information, the CoreSight topology, in the Component Connections tab.

6. Select the **Component Connections** tab.

   We must add a component connection detailing how the Cortex-A72 is connected to its associated CTI.

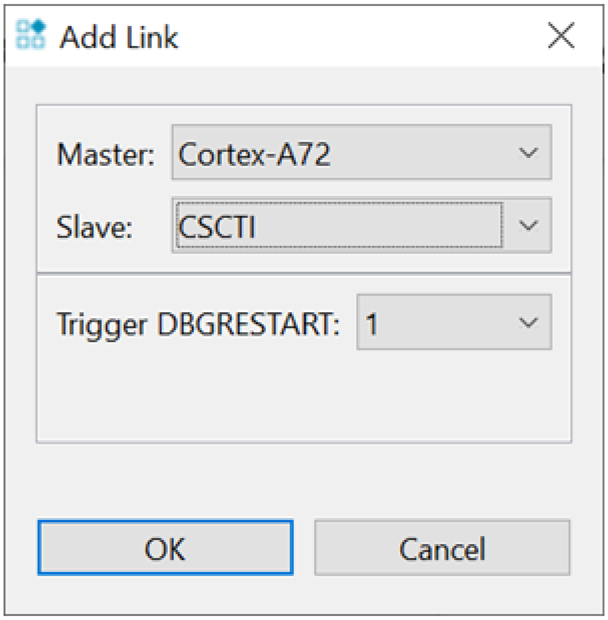
7. Click **Add Link**.

   The **Add Link** view lets you enter the connected **Master** and **Slave** components using the drop-downs and any additional connection details.

   In this case:

- The **Master** is the **Cortex-A72**.
- The **Slave** is the **CSCTI**.
- The **Trigger DBGRESTART** is **1**. **Trigger DBGRESTART** is the CTI channel that core start is connected to.

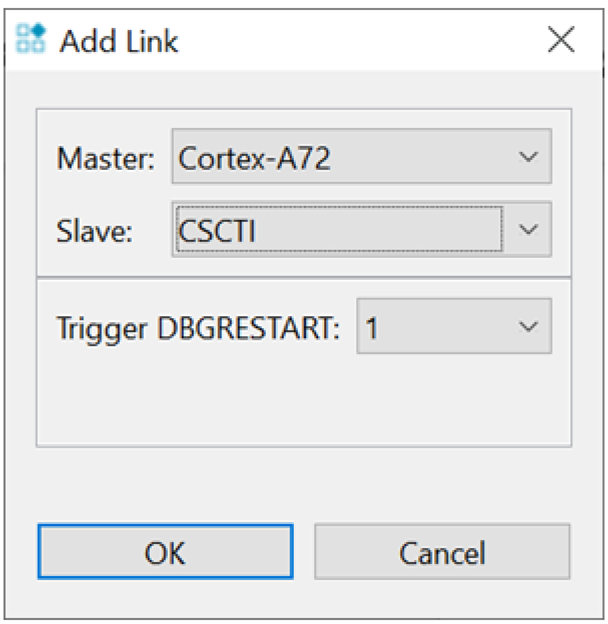
Figure 5-12: Adding a component connection between the Cortex-A72 and the CSCTI



8. Click OK.

   The created link appears in the Component Connections tab:

   Figure 5-13: Connection Component tab after CSCTI is added



   You now have all the information you to add the remaining debug-related devices and component connections to the platform configuration.

9. Add the devices and Component Connections for the devices listed in the table below.

Note

We have already added the devices or component connections highlighted below.

M: = Master

S: = Slave

<number>: = Trigger or Slave value

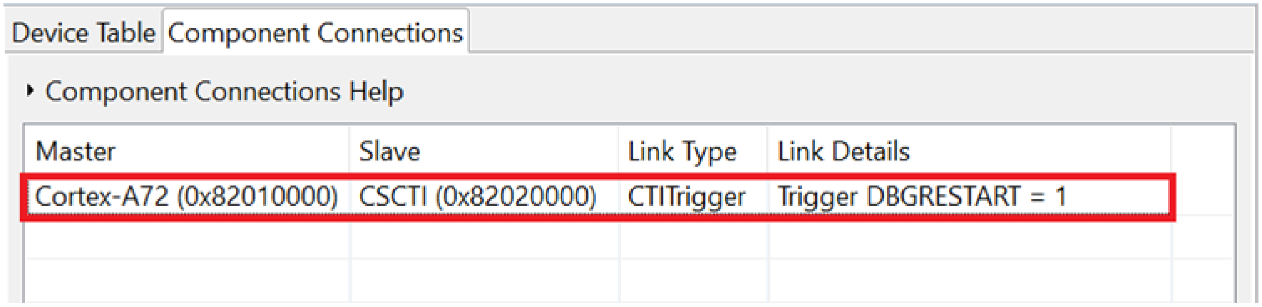
Device Type	PCE device name	AP index	CoreSight Base Address	Connected to
DP	ARMCS-DP_0	NA	NA	
AXI-AP	CSMEMAP (0: AXI-AP) - APv1	0	NA	ARMCS-DP_0
APB-AP	CSMEMAP (1: APB-AP) - APv1	1	NA	ARMCS-DP_0
Cortex-A72	Cortex-A72_0	1	`0x82010000`	S:1: CSCTI_0
CTI	CSCTI_0	1	`0x82020000`	-
Cortex-A72	Cortex-A72_1	1	`0x82110000`	S:1: CSCTI_1
CTI	CSCTI_1	1	`0x82120000`	-
Cortex-A53	Cortex-A53_0	1	`0x83010000`	S:1: CSCTI_2
CTI	CSCTI_2	1	`0x83020000`	-
Cortex-A53	Cortex-A53_1	1	`0x83110000`	S:1: CSCTI_3
CTI	CSCTI_3	1	`0x83120000`	-
Cortex-A53	Cortex-A53_2	1	`0x83210000`	S:1: CSCTI_4
CTI	CSCTI_4	1	`0x83220000`	-
Cortex-A53	Cortex-A53_3	1	`0x83310000`	S:1: CSCTI_5
CTI	CSCTI_5	1	`0x83320000`	-
AHB-AP-M	CSMEMAP (2: AHB-AP-M) - APv1	2	NA	ARMCS-DP_0
Cortex-M3	Cortex-M3	2	NA	S:7: CSCTI_6
CTI	CSCTI_6	2	`0xE0044000`	-

10. Click Save.

    PCE automatically builds the platform configuration

When complete, the Device Table is:

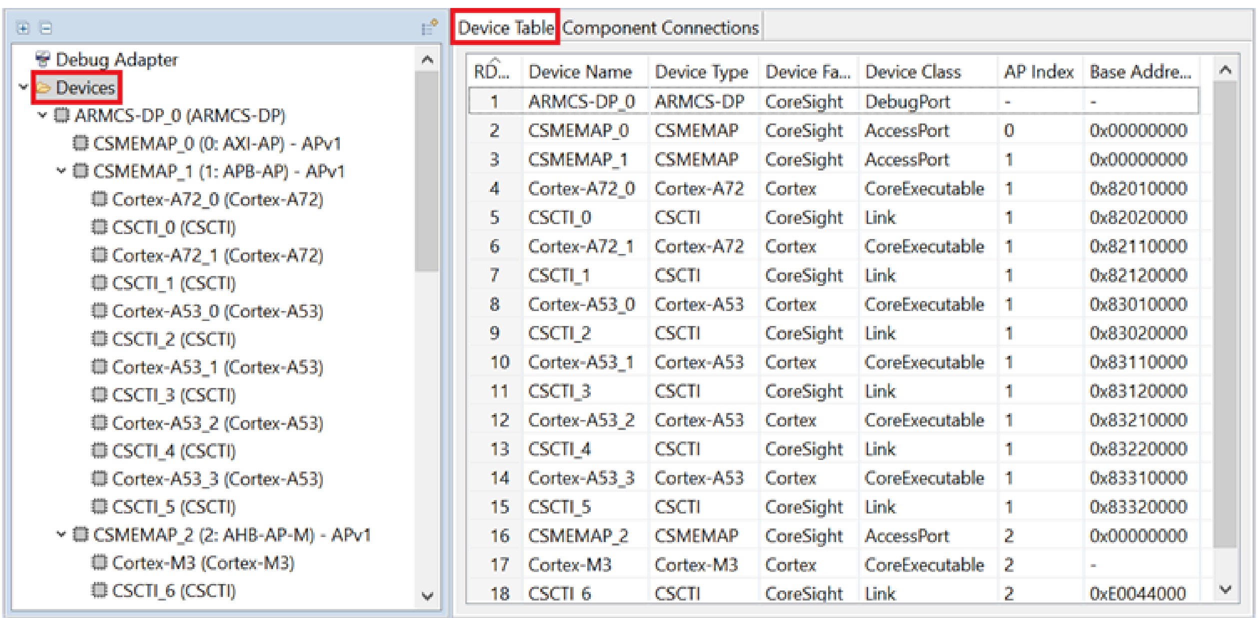
Figure 5-14: Complete debug Devices tab



Master	Slave	Link Type	Link Details
Cortex-A72 (0x82010000)	CSCTI (0x82020000)	CTITrigger	Trigger DBGRESTART = 1

The complete debug Component Connections is:

Figure 5-15: Complete Component Connections tab



RD...	Device Name	Device Type	Device Fa...	Device Class	AP Index	Base Addre...
1	ARMCS-DP_0	ARMCS-DP	CoreSight	DebugPort	-	-
2	CSMEMAP_0	CSMEMAP	CoreSight	AccessPort	0	0x00000000
3	CSMEMAP_1	CSMEMAP	CoreSight	AccessPort	1	0x00000000
4	Cortex-A72_0	Cortex-A72	Cortex	CoreExecutable	1	0x82010000
5	CSCTI_0	CSCTI	CoreSight	Link	1	0x82020000
6	Cortex-A72_1	Cortex-A72	Cortex	CoreExecutable	1	0x82110000
7	CSCTI_1	CSCTI	CoreSight	Link	1	0x82120000
8	Cortex-A53_0	Cortex-A53	Cortex	CoreExecutable	1	0x83010000
9	CSCTI_2	CSCTI	CoreSight	Link	1	0x83020000
10	Cortex-A53_1	Cortex-A53	Cortex	CoreExecutable	1	0x83110000
11	CSCTI_3	CSCTI	CoreSight	Link	1	0x83120000
12	Cortex-A53_2	Cortex-A53	Cortex	CoreExecutable	1	0x83210000
13	CSCTI_4	CSCTI	CoreSight	Link	1	0x83220000
14	Cortex-A53_3	Cortex-A53	Cortex	CoreExecutable	1	0x83310000
15	CSCTI_5	CSCTI	CoreSight	Link	1	0x83320000
16	CSMEMAP_2	CSMEMAP	CoreSight	AccessPort	2	0x00000000
17	Cortex-M3	Cortex-M3	Cortex	CoreExecutable	2	-
18	CSCTI_6	CSCTI	CoreSight	Link	2	0xE0044000

Understanding the platform configuration debug activities

If the build is successful, you can see which debug activities you can perform with the platform configuration.

1. Click Debug Activities.

   There are two main types of debug activity:

   - Bare Metal Debug
     - For debugging bare-metal environments such as non-OS boot code, firmware, and test cases.
   - Linux Kernel and/or Device Driver Debug
     - For debugging the Linux kernel or Linux kernel device drivers and applications.

The different activities let you connect to:

- Individual cores (denoted by Cortex-X_<number>) when X is the core type and number is the core number.
    - Debugger starting and stopping only starts or stops the individual core, not the rest of the cores.

Symmetric Multiprocessing core sets (denoted by SMP or big.LITTLE).

- Debugger starting and stopping starts and stops all the cores which are part of the SMP connection.For example, starting or stopping any core in a Cortex-A72 and Cortex-A53 big.LITTLE connection starts and stops the two Cortex-A72s and the 4 Cortex-A53s.

Test the debug aspects of the platform configuration

Test the platform configuration in the Development Studio perspective.

To make sure the platform configuration is working as expected, test the following:

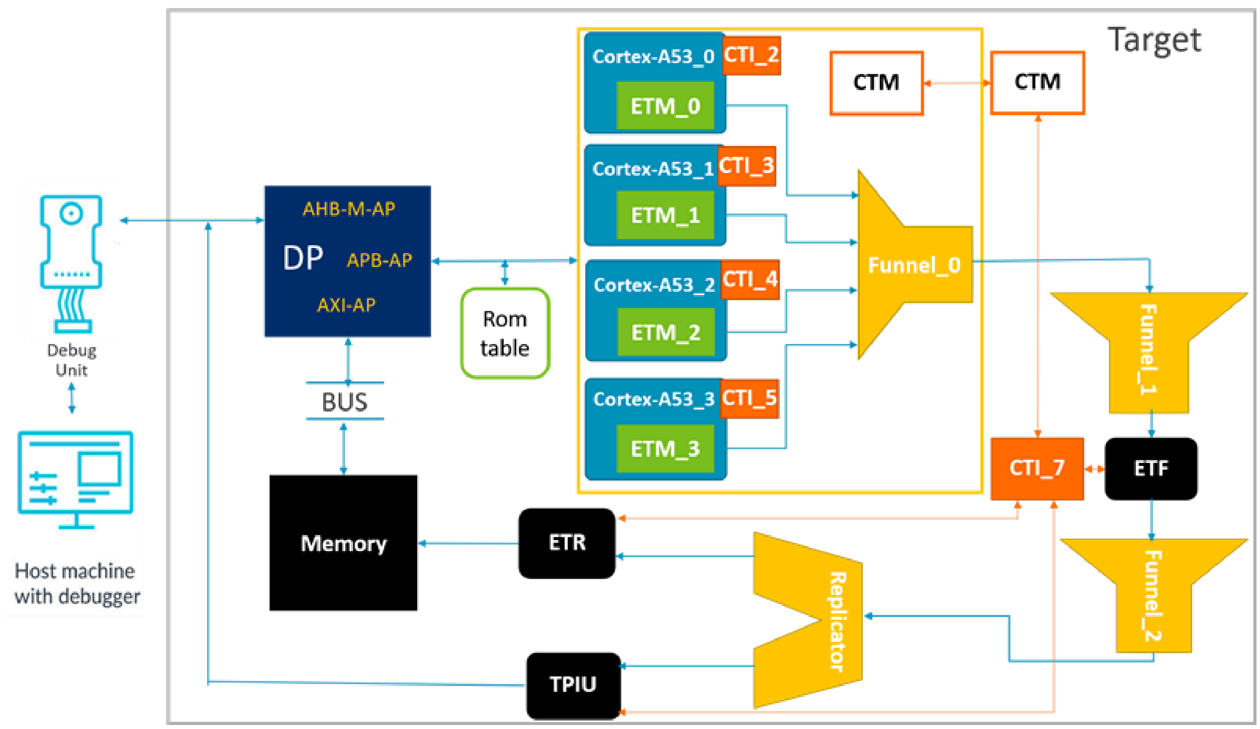
- Whether you can connect to and debug (for instance, stop and start) all the individuals cores using Bare Metal Debug.
- Whether you can connect to and debug (for instance, stop and start) all the SMP core sets using Bare Metal Debug.

6. Manually configuring a platform configuration for trace

In this section, we add the trace devices associated with the Cortex-A53 cluster, so that we can capture trace data for this cluster's cores.

Below is a block diagram of the trace-related devices for the target's Cortex-A53 cluster we are manually configuring:

Figure 6-1: Diagram of the example target's trace infrastructure



The connections between the Cortex-A53 associated CTIs and the Cortex-A53 cluster CTM are not present to make the diagram clearer.

The trace data flow is:

1. Each ETM generates trace data for its associated Cortex-A53 core.
2. The generated trace data combines in one trace stream by two funnels, Funnel_0 and Funnel_1.
3. The single trace stream is passed through an Embedded Trace FIFO (ETF).
4. The ETF output is passed into another funnel, Funnel_2.

5. The last funnel passes the single trace stream into a replicator to generate two identical trace streams.
6. Each trace stream is passed to an TMC ETR to store the trace data into target memory and a TPIU to export the trace data off the target to a debug unit.

There is also a CTI, CTI_7, connected between the ETF, ETR, and TPIU and the CTM for the Cortex-A53 cluster which allows these trace components to stop the cores in the cluster under specific circumstances.

We now start adding the trace devices to the platform configuration.

Add trace devices and component connections to the platform configuration

In this section, we add the below to the platform configuration:

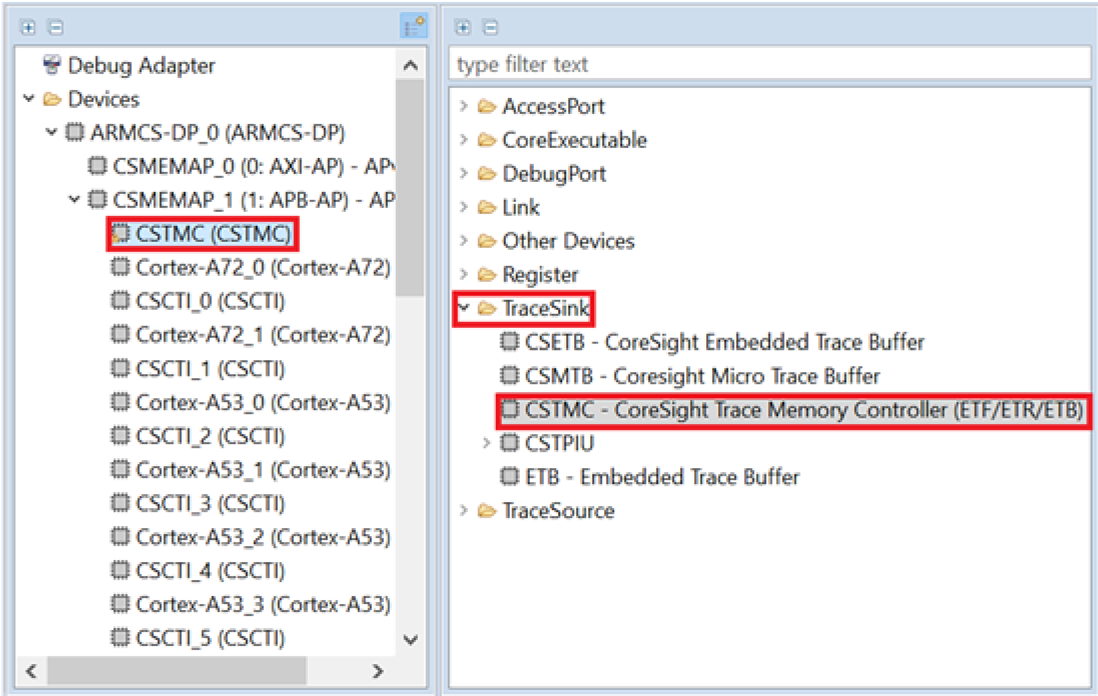
1. A Trace Memory Controller (TMC) ETF.
2. An ETM.
3. A component connection between the Cortex-A53 and the ETM.
4. A funnel.
5. A component connection between the ETM and the funnel.
6. The rest of the trace-related devices and component connections.

This exercise also covers good tests to check whether the platform configuration allows you to trace the target with the Arm Debugger.

1. Add a TMC (ETF).

   Add a CSTMC to the CSMEMAP_1:

Figure 6-2: Adding a CSTMC ETF



2. Click on CSTMC.

There are certain TMC settings which must be correct for Arm Debugger to use the TMC.

- CORESIGHT_BASE_ADDRESS
  - The lower 32-bits of the CoreSight base address for the TMC.
- CORESIGHT_BASE_ADDRESS_MSW
  - The higher 32-bits of the CoreSight base address for the TMC.
- CONFIG_TYPE
  - The type the TMC is configured for.The choices are ETF, ETR, and Embedded Trace Buffer (ETB).
- MEM_WIDTH
  - The width of the AMBA Trace Bus (ATB) into the ETF in bits.
- *RAM_SIZE_BYTES
  - The size of the ETF RAM in bytes.

Note

* means the device information entry appears after the CONFIG_TYPE is set.

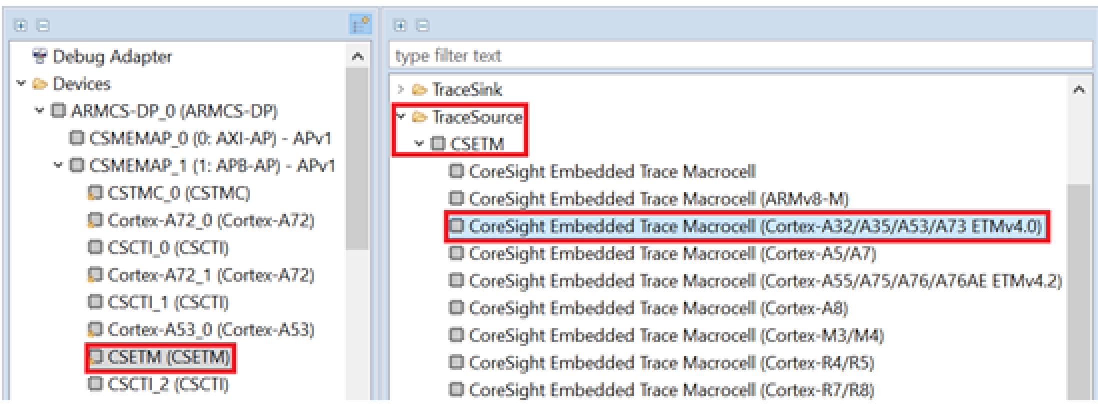
For our board, the details for the TMC are:

- CORESIGHT_BASE_ADDRESS is `0x80010000`.
- CORESIGHT_BASE_ADDRESS_MSW is `0x0`.
- CONFIG_TYPE is ETF.
- Leave RAM_SIZE_BYTES and MEM_WIDTH at the default values.

3. Add an ETM to Cortex-A53_0.

   Add a CSETM for the Cortex-A53_0:

   Figure 6-3: Adding an ETM



4. Click on CSETM.

   There are certain ETM settings which must be correct for Arm Debugger to use the ETM.

   - CORESIGHT_BASE_ADDRESS
     - The lower 32-bits of the CoreSight base address for the ETM.
   - CORESIGHT_BASE_ADDRESS_MSW
     - The higher 32-bits of the CoreSight base address for the ETM.
   - SUPPORTS_DATA_ADDRESS_TRACE
     - Whether the ETM supports data tracing.

   For our board, the details for the first Cortex-A53 ETM are:

   - CORESIGHT_BASE_ADDRESS is `0x83040000`.
   - CORESIGHT_BASE_ADDRESS_MSW is`0x0`.
   - SUPPORTS_DATA_ADDRESS_TRACE is `False`.

5. Add a Component Connection between Cortex-A53_0 and CSETM.
6. Add a CSTFunnel for the Cortex-A53 cluster and setup the CSTFunnel.

   For our board, the details for the Cortex_A53 cluster funnel are:

   - CORESIGHT_BASE_ADDRESS is `0x803C0000`.
   - CORESIGHT_BASE_ADDRESS_MSW is `0x0`.

7. Add a Component Connection between the CSETM and the CSTFunnel on Slave Interface 0.

   You now have all the information you to add the remaining trace-related devices and component connections to the platform configuration.

8. Add the devices and Component Connections for the devices listed in the table below.

Note

We have already added the devices or component connections highlighted below.

M: = Master

S: = Slave

<number>: = Trigger or Slave value

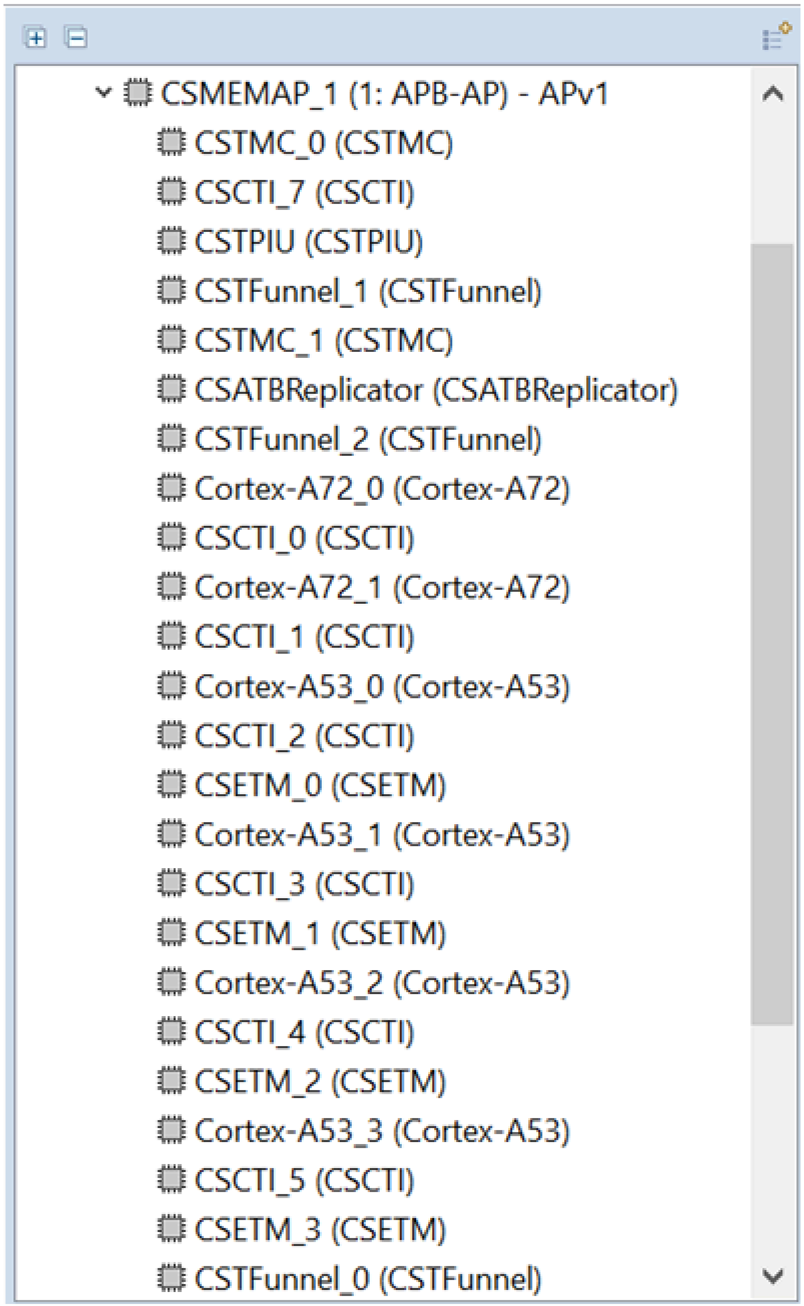
Device Type	PCE device name	AP index	CoreSight Base Address	Connected to
TMC (ETF)	CSTMC_0	1	`0x80010000`	S:0: CSTFunnel_2
CTI	CSCTI_7	1	`0x80020000`	S:0: CSTMC_0 S:1: CSTMC_1 S:3: CSTPIU
TPIU	CSTPIU	1	`0x80030000`	
Funnel	CSTFunnel_1	1	`0x80040000`	S: CSTMC_0
TMC (ETR)	CSTMC_2	1	`0x80070000`	
Replicator	CSATBReplicator	1	`0x80120000`	S:0: CSTPIU S:1: CSTMC_1
Funnel	CSTFunnel_2	1	`0x80150000`	S: CSATBReplicator
ETM	CSETM_0	1	`0x83040000`	S:0: CSTFunnel_0
ETM	CSETM_1	1	`0x83140000`	S:1: CSTFunnel_0
ETM	CSETM_2	1	`0x83240000`	S:2: CSTFunnel_0
ETM	CSETM_3	1	`0x83340000`	S:3: CSTFunnel_0
Funnel	CSTFunnel_0	1	`0x830C0000`	S:0: CSTFunnel_1

9. Click Save.

PCE automatically builds the platform configuration.

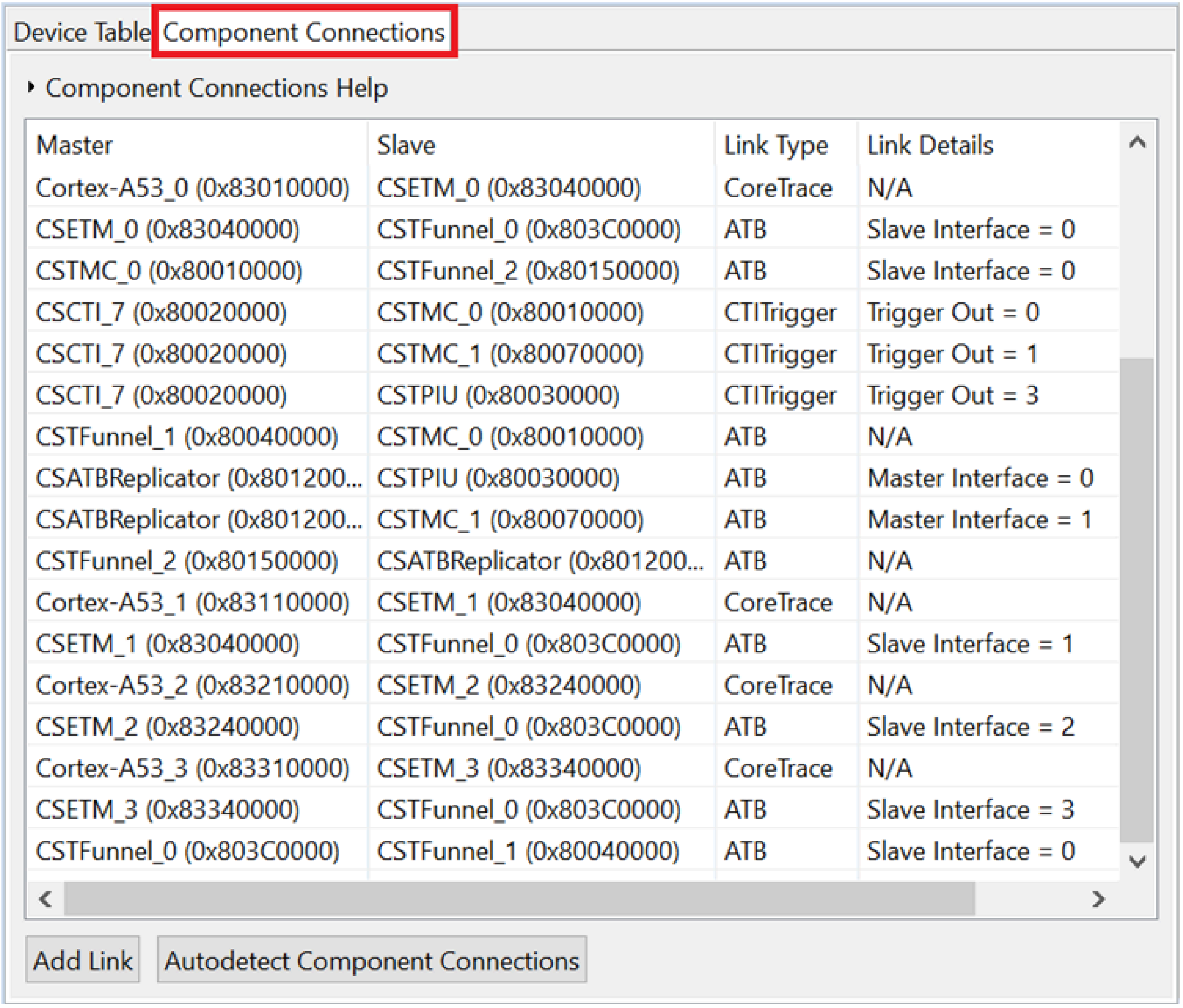
When complete, Devices is:

Figure 6-4: Complete trace Devices tab



The complete trace Component Connections is:

Figure 6-5: Complete trace Component Connections tab



Master	Slave	Link Type	Link Details
Cortex-A53_0 (0x83010000)	CSETM_0 (0x83040000)	CoreTrace	N/A
CSETM_0 (0x83040000)	CSTFunnel_0 (0x803C0000)	ATB	Slave Interface = 0
CSTMC_0 (0x80010000)	CSTFunnel_2 (0x80150000)	ATB	Slave Interface = 0
CSCTI_7 (0x80020000)	CSTMC_0 (0x80010000)	CTITrigger	Trigger Out = 0
CSCTI_7 (0x80020000)	CSTMC_1 (0x80070000)	CTITrigger	Trigger Out = 1
CSCTI_7 (0x80020000)	CSTPIU (0x80030000)	CTITrigger	Trigger Out = 3
CSTFunnel_1 (0x80040000)	CSTMC_0 (0x80010000)	ATB	N/A
CSATBReplicator (0x801200...	CSTPIU (0x80030000)	ATB	Master Interface = 0
CSATBReplicator (0x801200...	CSTMC_1 (0x80070000)	ATB	Master Interface = 1
CSTFunnel_2 (0x80150000)	CSATBReplicator (0x801200...	ATB	N/A
Cortex-A53_1 (0x83110000)	CSETM_1 (0x83040000)	CoreTrace	N/A
CSETM_1 (0x83040000)	CSTFunnel_0 (0x803C0000)	ATB	Slave Interface = 1
Cortex-A53_2 (0x83210000)	CSETM_2 (0x83240000)	CoreTrace	N/A
CSETM_2 (0x83240000)	CSTFunnel_0 (0x803C0000)	ATB	Slave Interface = 2
Cortex-A53_3 (0x83310000)	CSETM_3 (0x83340000)	CoreTrace	N/A
CSETM_3 (0x83340000)	CSTFunnel_0 (0x803C0000)	ATB	Slave Interface = 3
CSTFunnel_0 (0x803C0000)	CSTFunnel_1 (0x80040000)	ATB	Slave Interface = 0

Test the trace aspects of the platform configuration

Test the platform configuration in the Development Studio perspective.

To make sure the platform configuration is working as expected, test the following:

- Make sure you can get trace data from each ETM using the ETR.
- Make sure you can get trace data from each ETM using the TPIU.

Note on making changes to the platform configuration outside the PCE GUI

When configuring or modifying some target platform configurations, you might be required to make changes directly to the SDF or DTSL (.py) files without going through the PCE GUI. If this is the case, you must rebuild the configDB and test the platform configuration before trying to connect to the target.

To rebuild the configDB, go to Window > Preferences > Arm DS > Configuration Database and click Rebuild database.

To test the platform configuration, in the Configuration Database dialog:

1. Click Test platforms….
2. Select the platform configuration.
3. Click OK.
4. Resolve any errors found.
5. Save the platform configuration.
6. Rebuild the configDB.
7. Repeat steps 1 - 7 until no errors remain.