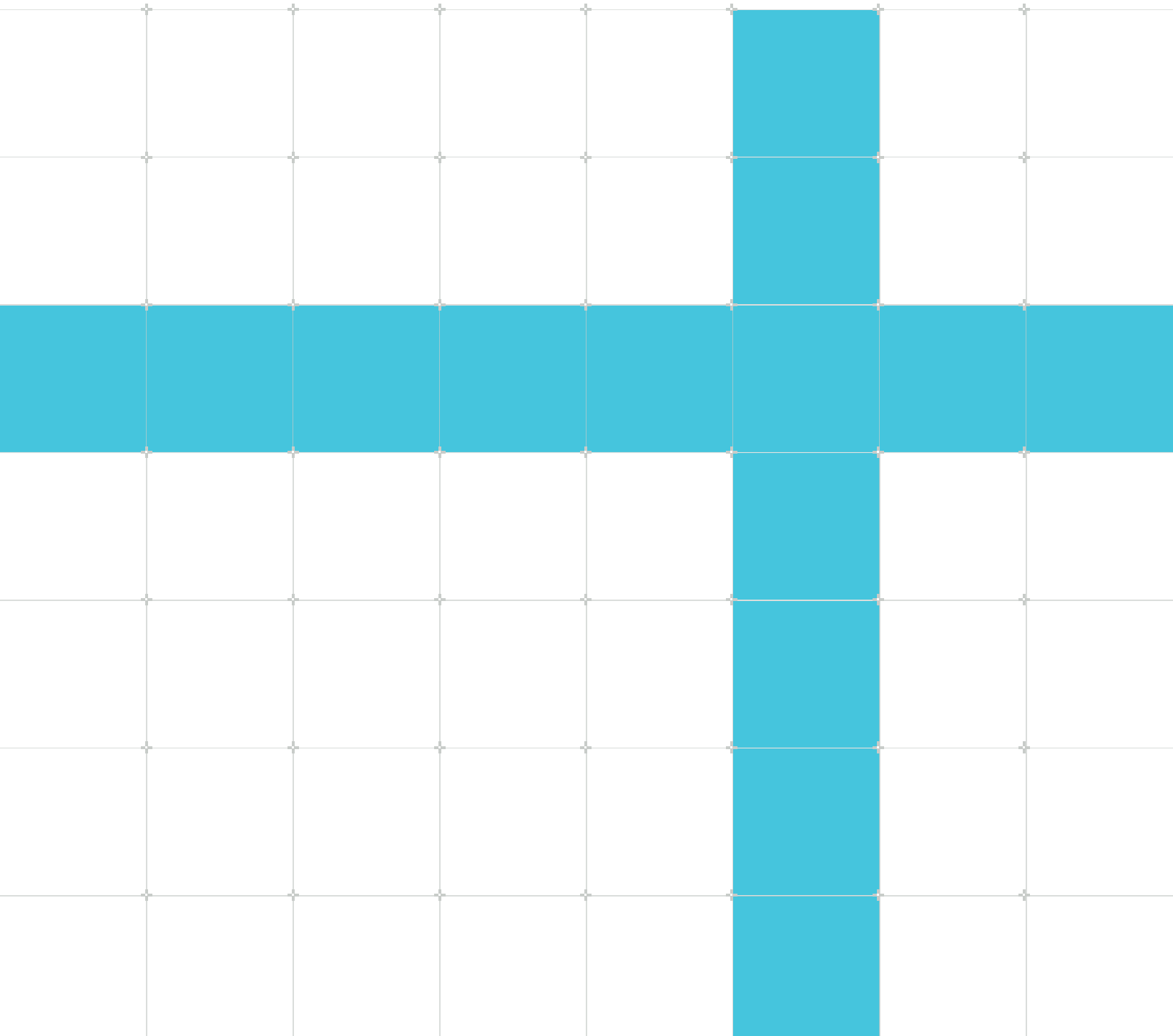
In depth analysis of autoconfiguring with PCE

Version 1.0

Issue 02
102701_0100_02_en



In depth analysis of autoconfiguring with PCE

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	10 November 2016	Non-Confidential	First version

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. Overview

This tutorial details on different functionality of the Platform Configuration Editor (PCE) allowing you to create suitable Debug Configurations for your custom targets.

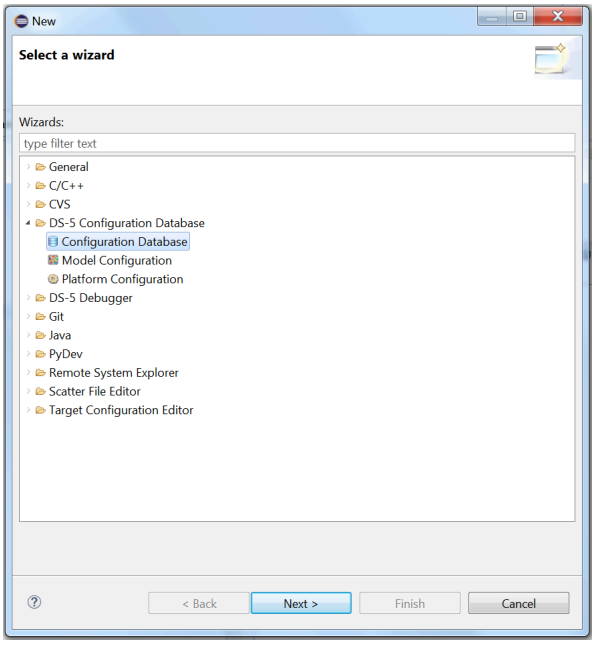
The last few years have shown an increase in the complexity of targets; these tend to mix several Arm architectures, include custom IP and have vast trace infrastructures. This change has had a considerable impact in the evolution of target bringup, the knowledge the software engineers must have and the specific requirements of the software tools needed for debug. The process of detecting the underlying platform components and topology needs to be as robust on a small single core design as on an elaborate server design with as much as a hundred cores. Arm's solution to this problem is the Platform Configuration Editor (PCE), tool which is integrated as part of the DS-5 Development Studio suite. In order to fully benefit from the content of this tutorial the user should have minimum understanding of CoreSight debug and trace architecture. To aid with this prerequisite, if required, some reading material has been provided in the last section of this tutorial.

2. Autoconfiguring your target

To autoconfigure your platform you require a DSTREAM unit and a working JTAG or SWD connection to your target. Lower cost debuggers from the ULINK family do not provide this functionality. You will also require an updated DS-5 version; as a minimum we suggest DS-5 v5.23.

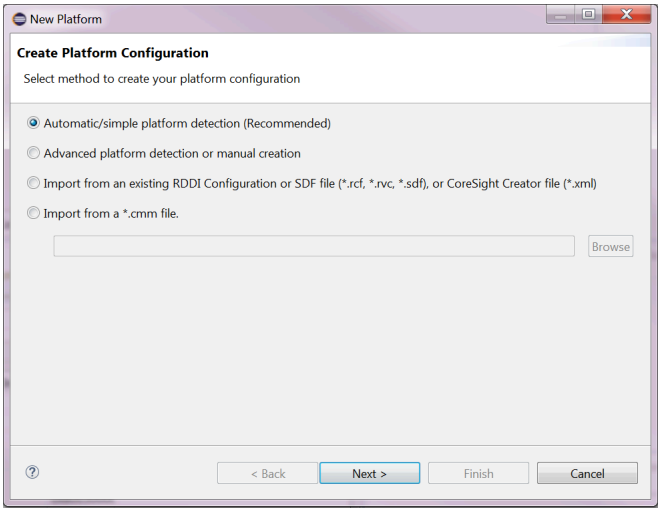
All Platform Configurations generated within PCE are grouped in Configuration Databases. Hence, if you do not already have a Configuration Database, the first step in autoconfiguring a new target and developing an associated Platform Configuration is to create a Configuration Database. To achieve this, click on **File** > **New** > **Other**... > **DS-5 Configuration Database** > **Configuration Database**.

Figure 2-1: A configuration database.



Choose a database name in the next view. Once created, this is where you will store as many Platform Configurations as you want. To start adding one, open the **Project Explorer view** (it will be placed on the left-hand side by default if DS-5 Configuration Perspective is used), right-click on the Configuration Database you have just created and select **New** > **Platform Configuration**.

Figure 2-2: Platform configuration options.

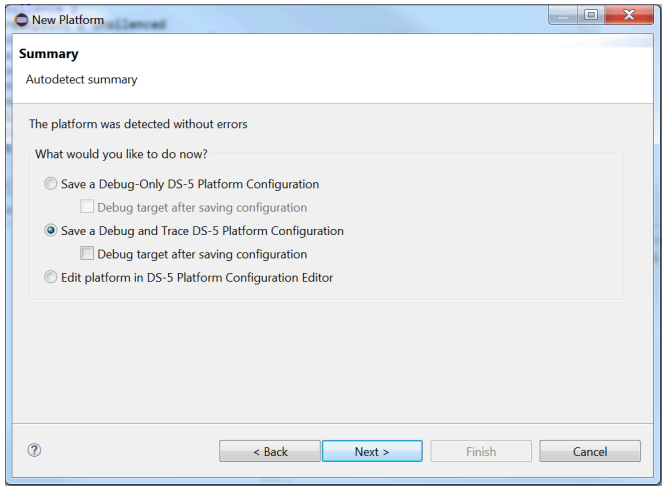


For autodetection select the **Automatic/simple platform detection** (Recommended) and then the **connection details** to the appropriate DSTREAM unit in the **Debug Adapter Connection view**. Clicking **Next** at this stage will start the autoconfiguring process, progress of which will be tracked at the lower side of the **New Platform window** and eventually as a complete summary in the Console window. The autoconfigure log will be described in the following section of this tutorial.

If the Summary shows `Target detection failed. Autodetection failed` that suggests that the DSTREAM unit has been unable to connect to the Debug Access Port (DAP) or the TAP controller. In this situation further investigations need to be performed; suggestions include trying to establish connection with CoreSight Access Tool (CSAT), double checking the design, or creating part of the configuration manually (the `.sdf` file). Section 3 from the What can be done when autoconfigure fails chapter will cover the last approach.

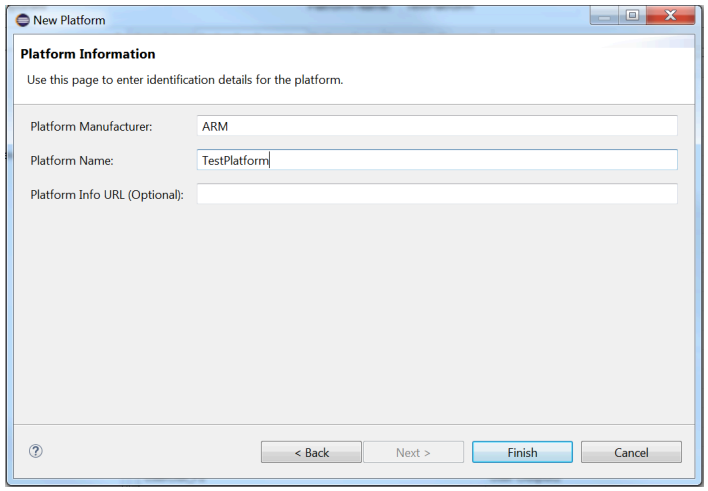
If the Summary window shows you options of saving the platform having been autodetected, similarly to the screenshot below, select the recommended **Save a Debug and Trace DS-5 Platform Configuration**.

Figure 2-3: Summary window for New Platform.



This will invoke the **Platform Information** window which requires you to add the Manufacturer and Name of the device.

Figure 2-4: Platform information for New Platform.



You now have a full Debug Configuration for your target which can be accessed from the **Project Explorer** window > <Name of the Configuration Database> > **Boards** > <Platform Manufacturer> > <Platform Name >. This will automatically be added to the list of available configuration databases shown in the **Debug Configuration** window. By creating a new Debug Configuration you are now able to select the target you have just autodetected and connect or debug an image.

In terms of the files contained in the Debug Configuration you can notice that component information is collected in an .sdf file. This is then used to generate DTSL scripts, debugger project settings and a configuration file for the DSTREAM (`.rcf`). The `.sdf` file does not only show component information but also links between these. If the information visible in the Device Table and Component Connections does not match with your expectations (i.e. your CoreSight topology

diagram and ROM table information), then manual changes could be done in this file. The PCE tool allows you to rebuild your target configuration following any changes made.

3. Interpreting the PCE log following autoconfigure

This section aims to give explanations of some of the content found in the autoconfiguration log that PCE produces. This information can then be used to identify any target or potentially tool issues which lead to incomplete description files.

The process of autoconfiguring aims to find:

- the number of devices on the platform,
- the CoreSight components on each device and finally,
- determine the connections between them.

Knowing the way JTAG works and its associated JTAG state machine is important when investigating the number of devices. Related to this, in order to enable communication with all of the devices, the debugger also needs to be aware of the DR (data register) and IR (instruction register) length for each of the devices on the scan chain. This is achieved by the PCE tool by originally putting all devices in bypass mode and then scanning 512 logical `1`s followed by 512 logical `0`s. This can be observed on the log below:

```
Counting devices:
DR Chain [1024]:
[00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
Device Count: 1
```

During the next step the tool triggers a TAP reset via Test-Logic-Reset state. Following this, the 32-bit IDCODE of the device is read, matching it against a predefined list and identifying the IR length of the component. The ARMCS-DP specified at the end of the log is actually the Debug Access Port (DAP):

```
Reading IDCODEs:
DR Chain [32]:
[01001011101000000000010001110111]
Device 0 has IDCODE = 0x4BA00477

IR Chain [1024]:
[11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
Device 0 has IR Length = 4

Getting device names:
Device 0 detected as ARMCS-DP
```

Failure at this point

One of the reasons for autoconfiguration failure is if the TAP does not reset into the standard IDCODE sequence or respond to a standard IDCODE IR value. If this is the case, then depending on how advanced you are in your project, you can either investigate why the IDCODE is not coming up as expected or you can manually add the device incapable of being detected.

Section 2 from the What can be done when autoconfigure fails chapter describes how this could be achieved.

Once the IDCODE is matched to the DAP of the device, connection with this is established. The tool then continues with identifying the Access Ports (AP) under that DAP. For example, the target can have:

```
Connected to DAP device

AP buses detected:
         AHB-AP
         APB-AP
         JTAG-AP
```

For each Access Port the tool will look for the ROM table base address found within the BASE, Debug Base Address register or any single components tied directly to the Access Port. The ROM table contains entries for each of the CoreSight components. Note that a JTAG-AP (the legacy AP) will not have a ROM table base address. An AXI-AP may have a ROM base address, however, an AXI-AP is typically used to directly access main system memory.

PCE will only utilise the value stored in the BASE, Debug Base Address register if bit [0] of the address is set to indicate that a ROM table is present.

Reading through the ROM table allows the tool to search for the CoreSight components (cores, trace source, trace sinks, CTIs, etc.) and the base address at which these are located. This enables PCE to perform topology detection on the device. Further decide information is then read; for example, an Embedded Trace Macrocell (ETM) might show:

```
Acquiring device info for CSETM (0x8203D000):

         ETM Version: 3.5
         Device supports timestamps
         Device supports context IDs
         Device supports cycle accurate trace
         Device supports data address trace
         Device supports data value trace
         Device supports data only mode
         Device supports trace range
```

Finally, the last section of the autodetection, depicted in the log, reads the links between the different CoreSight components. This becomes particularly important when testing trace, because the trace flow needs to be guided from the trace sources to the sinks, via some potential links. The correctness of these links for enabling trace is not limited to one particular type of trace. As an example, if your target supports off-chip trace then you might see the following links:

Processors to sources of trace:

```
MASTER = Cortex-A7 (0x82030000)      SLAVE = CSETM (0x8203C000)
MASTER = Cortex-A7 (0x82032000)      SLAVE = CSETM (0x8203D000)
```

```
MASTER = Cortex-A7 (0x82034000)     SLAVE = CSETM (0x8203E000)
```

Sources to the funnel collecting the trace stream:

```
MASTER = CSITM (0x80050000) (0)     SLAVE = CSTFunnel (0x80040000) (3)
MASTER = CSETM (0x8203C000) (0)     SLAVE = CSTFunnel (0x80040000) (2)
MASTER = CSETM (0x8203D000) (0)     SLAVE = CSTFunnel (0x80040000) (4)
MASTER = CSETM (0x8203E000) (0)     SLAVE = CSTFunnel (0x80040000) (5)
```

Funnel to trace sinks:

```
MASTER = CSTFunnel (0x80040000) (0)          SLAVE = CSETB (0x80010000) (0)
MASTER = CSTFunnel (0x80040000) (0)          SLAVE = CSTPIU (0x80030000) (0)
```

4. Controlling the autodetection settings

Sometimes the default settings used for performing autodetection are not appropriate with the design of your target. In these situations, there are a few parameters which you can change; these can be modified from the Autodetect tab after selecting “Advanced platform detection or manual creation”.

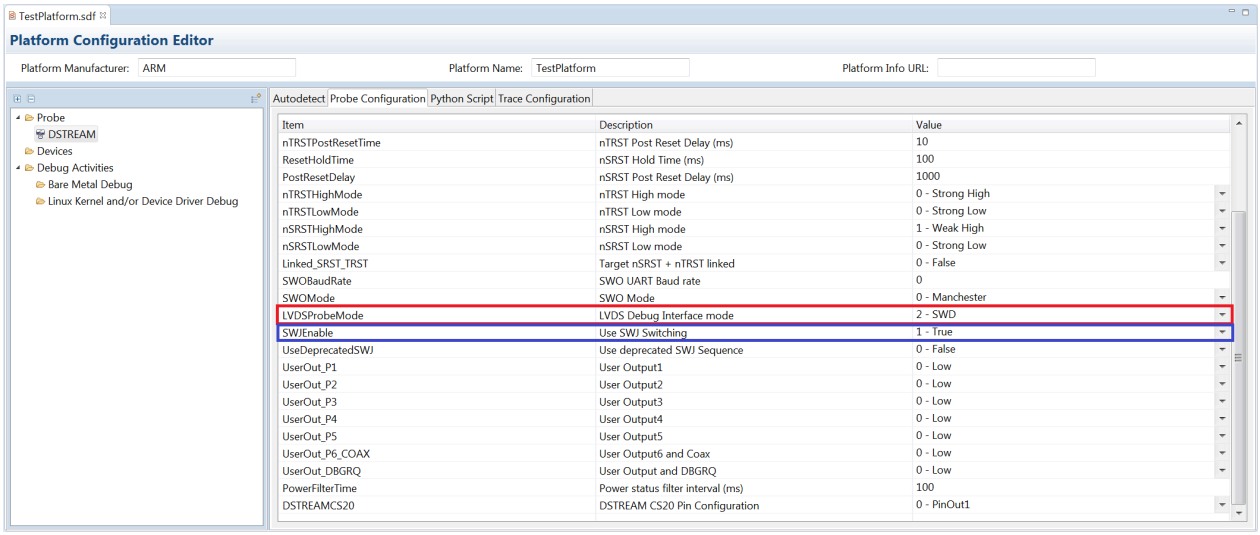
A classic example of setting you might want to change is the JTAG clock speed. By default PCE uses 10MHZ JTAG speed and potentially adaptive clock if detected. If the target is an FPGA or Emulation platform, the JTAG clock speed may need to be reduced.

Depending on how specific (mostly reset) signals are pulled on the target you might also want to modify **nTRSTHighMode** (set to strong high by default), **nTRSTLowMode** (set to strong low by default), **DoSoftTAPReset** (set to true by default, this is equivalent to setting TMS=1 for 5 TCKs within the JTAG TAP state machine) or **TResetOnInitConnect** (set to true by default). The user can find these in the **Probe Configuration** tab or the **Autodetect** > **Advanced Options**.

5. Platform detection using SWD

If your design supports Serial Wire Debug (SWD) then additional configurations need to be made to the default setup in PCE before the autoconfigure process is started. Instead of choosing **Automatic/simple platform detection** (Recommended) in the **New Platform Configuration** you will have to select **Advanced platform detection or manual creation**. The SWD specific settings that need to be done are setting **LVDSProbeMode** to 2 - SWD (red box in the screenshot) and, only if your target supports both JTAG and SWD then you must modify SWJEnable to be 1 - True (blue box in the screenshot). The latter option enables SWJ Switching, sequence which is sent in JTAG mode in order to inform the target that SWD is to be used.

Figure 5-1: Platform Configuration Editor.



To start autoconfiguring choose a Connection Address in the Autodetect tab and, depending on the target, modify the clock speed and then click on the **Autodetect Platform** button.

Support for multi-drop SWD has been added in DS-5 5.24 and its associated DSTREAM firmware. If your design includes multiple Debug Access Ports (DAP) using SWD and the debugger needs to switch between these please contact support@arm.com.

6. What can be done when autoconfigure fails

There are multiple reasons why the autoconfigure process can fail. Misconfigurations of the system, devices powered down, additional custom components interfering with the rest of the devices are only a few examples. The next few sections attempt to cover some of these areas and focus on the methods used to finalise the creation of the Platform Configuration.

1. Common Design Issues

   If the system is not in hard silicon yet then changes could potentially still be made to the design to ensure that your customers are able to successfully connect to Arm's debuggers. Some of the typical mistakes which hinder autodetection in PCE are related to incorrect power domain handshaking, incomplete ROM table and Access Ports setup or locking out accesses from the external debugger to the CoreSight components.

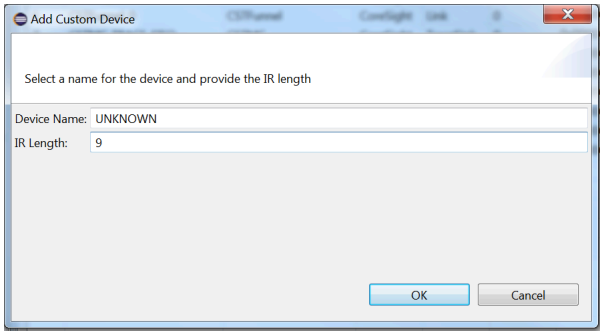
   Further advice on this topic is given in the First Time Chip Bring-up Success Application note available here.

2. Manually adding devices

   Some of the design flows mentioned in the section above cannot be fixed if the system is already in silicon. Similarly, sometimes the tool does not recognise the design and is incapable of autoconfiguring the target. PCE allows you to manually add either all the devices followed by autoconfiguring the components on each device or make up the entire system within an `.sdf` file. The rest of the files will then be built in the usual way, allowing you to have access to a complete Debug Configuration. This section describes the first method, which requires the least amount of work.

   For the purpose of this example let's assume the design has a custom component of IR Length 9, followed by a generic Debug Access Port(DAP) component. To add the first component right-click on the word **Devices** in the left hand side of the pane and select **Add Custom Device**. In the pop up window fill in the fields as shown below:

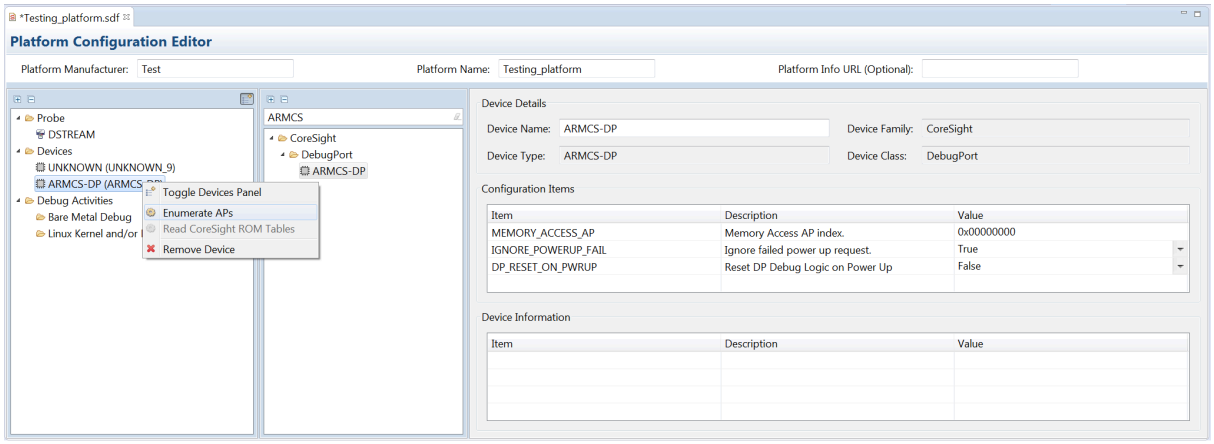
   Figure 6-1: Pop up window for Add Custom Device.



   The ARMCS-DP can then be added from the Devices Panel which is accessible from the left hand-side panel of the view by clicking on the **Toggle Devices Panel** button. Establish

connection to your DSTREAM unit, which in turn is connected to your target. This can be achieved by browsing for your DSTREAM unit via the **Probe** > **DSTREAM** > **Autodetect** > **Probe Connection tab**. If connection was established, then right-clicking on the **ARMCS-DP** should now show highlight the **Enumerate APs** option, as shown in the tree structure below:

Figure 6-2: Platform Configuration Editor for Enumerate APs.



If the Access Ports (APs) under the DAP have been correctly identified then you should be able to right-click on the port and select **Read CoreSight ROM tables** to fill in the components.


Note

PCE is unable to autodetect components on the JTAG-AP scan chains, which are normally used for Arm9/Arm11 cores. For these cases, you will have to manually add the components, as described in the next section.

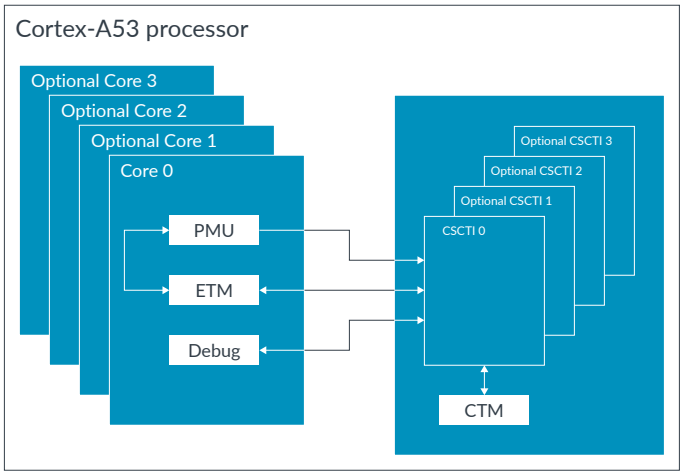
---

3. Complete manual creation of the `.sdf` file

   We have mentioned before that the last resort, when autoconfigure completely fails, is to manually create an .sdf file. This section will take you through this process, with the aid of an arbitrary example.

   Aim: Provide suitable Debug Configuration files for the design shown below

Figure 6-3: Cortex-53 processor.



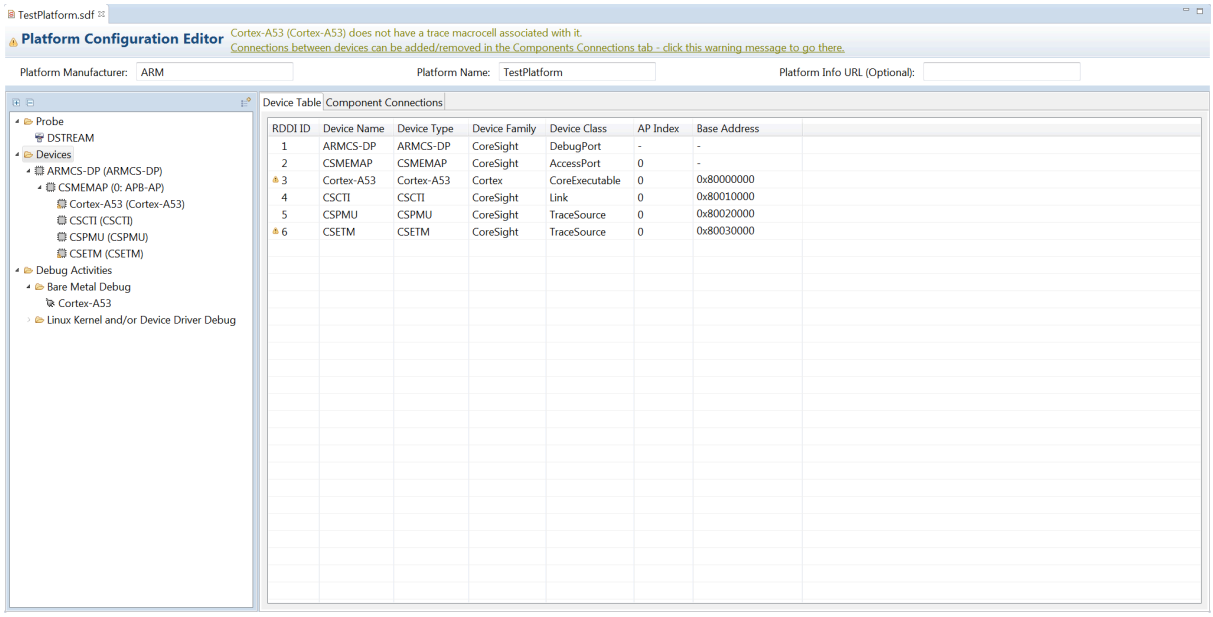
Select **New Platform Configuration** > **Advanced platform detection** or **manual creation**. On the next window fill in the name of the manufacturer and the platform name. Clicking the **Finish** button automatically creates an empty `<Platform_name>.sdf` file which we will have to populate with CoreSight components.

In the left hand-side panel of the view click on the **Toggle Devices Panel** button; this should open up a list of components in the middle of the view. Populate the Devices present in the left-hand pane, by selecting the appropriate components and drag and dropping, with the following components:

- **CoreSight** > **DebugPort** > **ARMCS-DP**
- **CoreSight** > **AccessPort** > **CSMEMAP (APB-AP)**
- **Cortex** > **CoreExecutable** > **Cortex-A53. This needs to be placed under the AP 0: APB-AP**
- **CoreSight** > **TraceSource** > **CoreSight Embedded Trace microcell (ETMv4.0) generates CSETM**
- **CoreSight** > **CSCTI** > **CoreSight Cross Trigger Interface generates CSCTI**

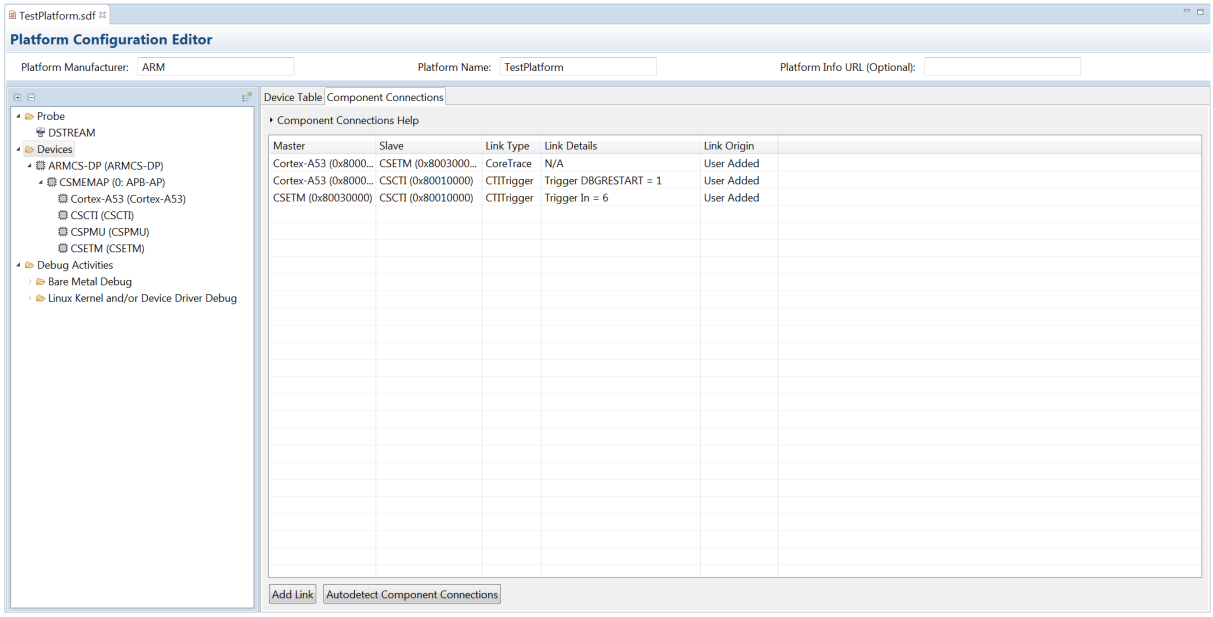
Specify base addresses for all components by clicking on the component and modifying the `CORESIGHT_BASE_ADDRESS` value in the Configuration Items area. The Device Table window should now look similarly to the screenshot below. Since no links between components have yet been created the tool has marked this as an incorrect topology by showing a couple of errors at the top of the window.

Figure 6-4: Platform Configuration Editor Device Table.



Once we have added all components to our list we then need to specify the topology. The way the CoreSight components are connected between each other defines the links between the trace sources and the trace sinks or the Cross Trigger Interface. The diagram shown at the beginning of this example should contain all the information required. Click on **Devices** in the left hand side pane, select the Component Connections tab and then using the Add Link button create the necessary connections. For this particular example we will require three entries, as shown in the screenshot below:

Figure 6-5: Platform Configuration Editor Componenent Connections.



Save the `.sdf` file and then, similarly to other configurations created by autoconfiguring, build the project by right clicking on the **Platform name** and selecting **Build Platform**. The Debug Configuration files for the platform can now be tested.

4. Have the CoreSight SoC-400 Integration tests been run?

   A lot of the times when the tools are unable to autodetect the CoreSight components and the existing topology this is related to the original design, configuration of registers and even stitching of components. For this reason, it is worth discussing with your design team early in the development cycle if the system has been fully validated, preferably in simulation. CoreSight SoC-400 provides with a number of Integration Tests which can be run in order to check your debug and trace elements. This is fully detailed in the Arm CoreSight SoC-400 User Guide.

   As an example, the discovery.c test performs a ROM table walk for each Access Port (AP) in the SoC and displays the discovered CoreSight components used in the debug and trace subsystem. This is very similar to the way in which PCE operates; so if the discovery test fails then it is guaranteed that the autoconfigure will not be able to identify all components.

   Often mistakes are done in the design when configuring the trace infrastructure. To test the data flow from the potentially multiple Embedded Trace Macrocell (ETM) or Program Flow Trace (PTM) all the way to the Embedded Trace Buffer (ETB) or Trace Port Interface Unit (TPIU) for off-chip trace you can use the trace.c integration test. This executes predefined trace stimulus sequences for each of the trace sources and then checks the output received at each trace sink against a golden-reference. If the trace test shows an incorrect stream of data coming out, then we suggest that waveforms are carefully checked in simulation, before moving on to silicon and failing to see any trace displayed in the debugger.

   Similar tests are available for the Cross Triger Interfaces (CTI) or the availability to halt the processors via the CTIs, details of which are outside of the scope of this tutorial.

7. Differences when moving from Debug Hardware Configuration to PCE

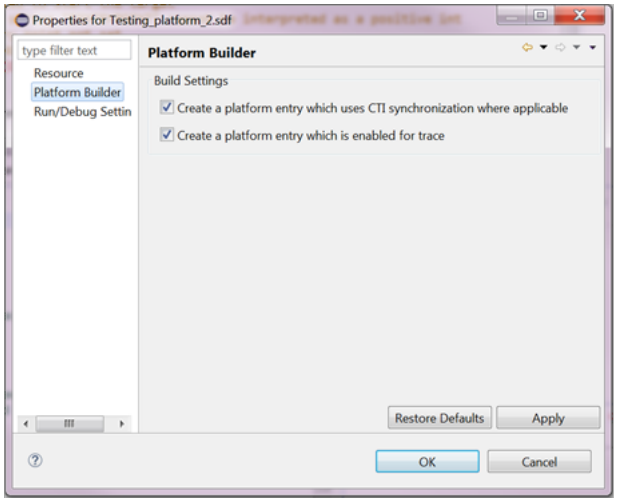
For customers who used the earlier versions of DS-5, they would probably be used to creating `.rvc` files using the Debug Hardware Configuration tool and then feeding the result to another tool called cdbimporter, which would generate the entire configuration. With the complexity of the targets increasing over the years, the functionality of this product had to be improved, hence the decision of designing a completely new DS-5 integrated tool. One of the main reasons for which this required a complete refresh was because when using Debug Hardware Configuration assumptions were made about the topology of the system. As an example, for complicated trace infrastructure requiring multiple trace buffers, replicators and funnels, the tool would use some hardcoded assumptions to compile the full design. Most of the times these assumptions were correct. However, sometimes possible design issues, such as not marking the **is present bit** for the **ROM Table Base Address**, were not uncovered when autodetecting. But most importantly the DTSL script had to be manually edited to allow for the full system to be debugged and traced, thus compromising the whole idea of autoconfiguring. For this particular example, this issue is not seen with PCE since this uses integration registers to define the topology of the system.

8. How to test your Debug Configuration files

It is very important to understand that once created the Debug Configuration files need to be tested in a progressive manner in order to efficiently track any inconsistencies. For example, testing off-chip trace before basic connection to the target is established is not the way to go. The flow described below is not an exhaustive list of features you need to test, but should give the user a starting point when bringing up the platform.

1. Test basic connection between the debugger and the target This assumes that if using a DSTREAM unit the TARGET LED on the DSTREAM unit and the VTREFA/VTREFB LED on the DSTREAM probe are lit up. Configure the Debug Configuration for Bare Metal Debug involving only one of the cores (eg: `Debug Cortex-A53_0`), with no image being loaded (Files tab does not link to an image), and **Connect only** selected in the **Debugger tab** > **Run control**.
2. Load an image onto the target and test basic debug An image can be loaded from the **Files** > **Application on host to download menu** and then the entry point can be set from the **Debugger** tab > **Debug from entry point/Debug from symbol** (main). A log of the commands performed by the debugger is shown in the **Commands** window; this could be useful, for example, in situations where the connection was successful but the image could not be loaded at the specific address. If the image loads successfully carry on by pressing the **Continue** button and testing interrupting the execution.
3. Test more advanced debug features There are a multitude of features that could be tested, but as a minimum we suggest source code and disassembly instruction stepping, hardware and software breakpoints to be set, values in memory and core/system registers to be viewed and modified. To test this functionality either the specific views (Memory, Disassembly, Registers, Breakpoints, etc.) can be used, or commands can be written in the Commands view (eg: memory set, hbreak, etc.).
4. Try data/instruction tracing This assumes the target includes specific trace support and the Debug Configuration has not been built with Debug only support. To note, if trace support is not required for the platform this can be controlled via the build options by right clicking on the `*.sdf` file and selecting **Properties** > **Platform Builder** and unticking **Create a platform entry which is enabled for trace**.

Figure 8-1: Properties for the Platform Builder.



Trace should be tested systematically, core by core and separately for each type of trace (eg. On chip Trace Buffer (ETB), DSTREAM 4GB Trace Buffer, etc.). All the necessary settings appear in the DTSL Options Edit... view accessible from the Connection tab. Apart from the support in the debugger itself, the LED indicators on both the DSTREAM unit and probe, such as the TRACING, DATA, TRC CLK could give good indicators if the target is tracing.

Note

The aim is to get a green TRC CLK LED on both the DSTREAM probe and the DSTREAM unit; this suggests a valid trace clock.

---

5. Linux Kernel and/or Device Driver Debug An officially supported kernel version could be used to test Kernel debug (available from the release notes of DS-5).

9. Next steps

This tutorial has provided you with a description of the different functionalities of the Platform Configuration Editor. Creating a suitable Debug Configuration for your custom target can be straightforward if you consider some of the inbuilt flexibility of the product. This can get around proprietary components, minor RTL defects and complex heterogeneous devices.

10. Related information

Aside from the official DS-5 documentation available here also see the First Time Bring-up Success Application Note. For customers who require more knowledge about CoreSight we also suggest going through the following material:

- CoreSight Technical Introduction White Paper
- Arm Debug Interface Architecture Specification ADIv5.0 to ADIv5.2
- Arm CoreSight SoC-400 Technical Reference Manual