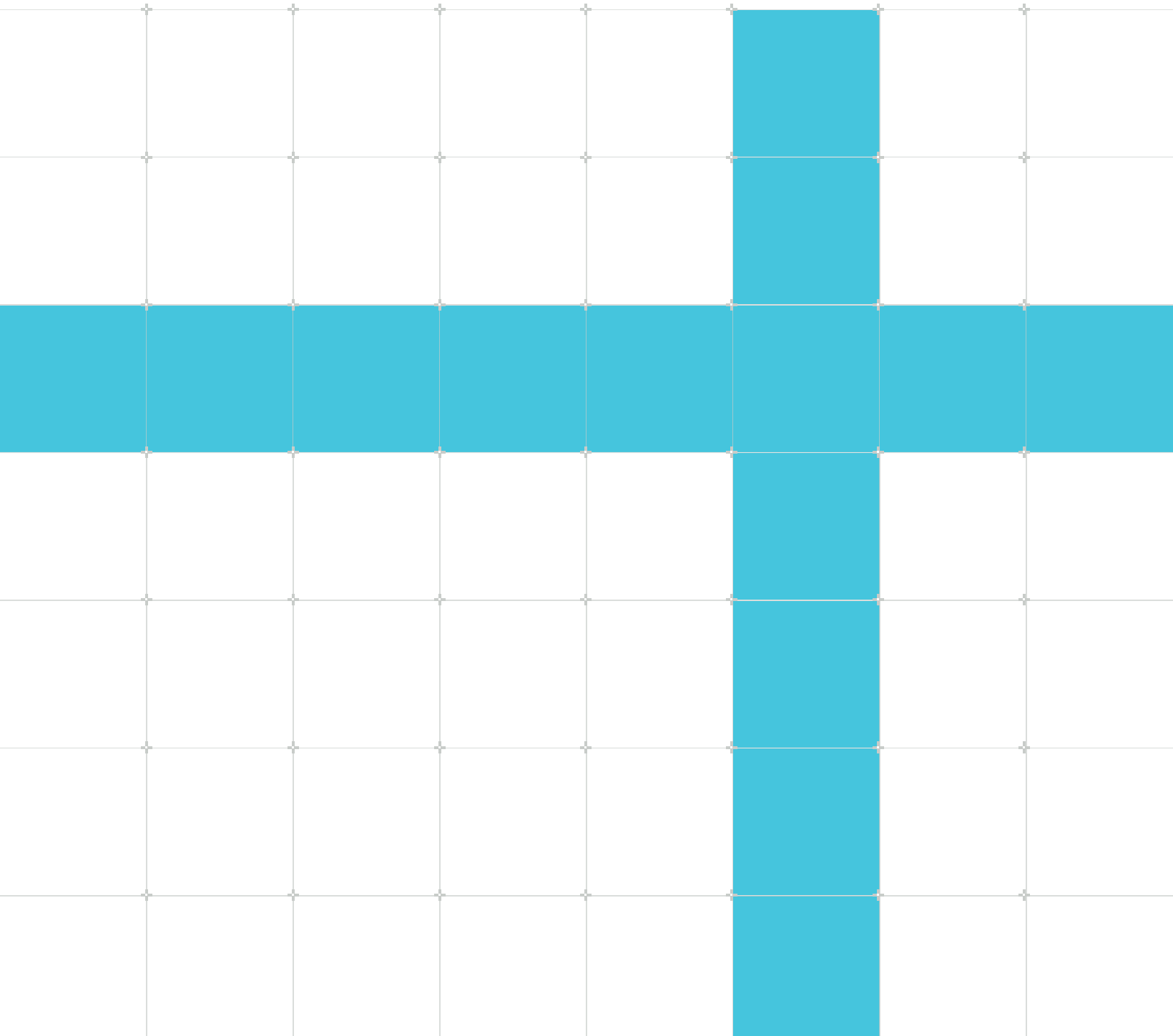
How PCE identifies the CoreSight components on the target board

Version 1.0

Issue 01
102582_0100_01_en



How PCE identifies the CoreSight components on the target board

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	1 January 2021	Non-Confidential	First release

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction..... 6

2. Arm SoC debug and trace..... 7

3. How is PCE useful?..... 8

4. How does PCE detect the information?..... 9

5. Related information..... 24

1. Introduction

This tutorial shows how Platform Configuration Editor (PCE) works to detect the underlying system architecture of an SoC. You will learn about the entire PCE process, how to read the log generated by PCE, and understand the common messages that occur in the PCE process.

2. Arm SoC debug and trace

Arm CoreSight is a collection of components that can be used to debug and trace an Arm SoC. Arm has CoreSight SoC-400 and CoreSight SoC-600 solution for debug and trace of complex SoCs. CoreSight SoC-400 is an Arm debug interface architecture v5(ADIv5) compliance system and CoreSight SoC-600 is an Arm debug interface architecture v6(ADIv6) compliance system.

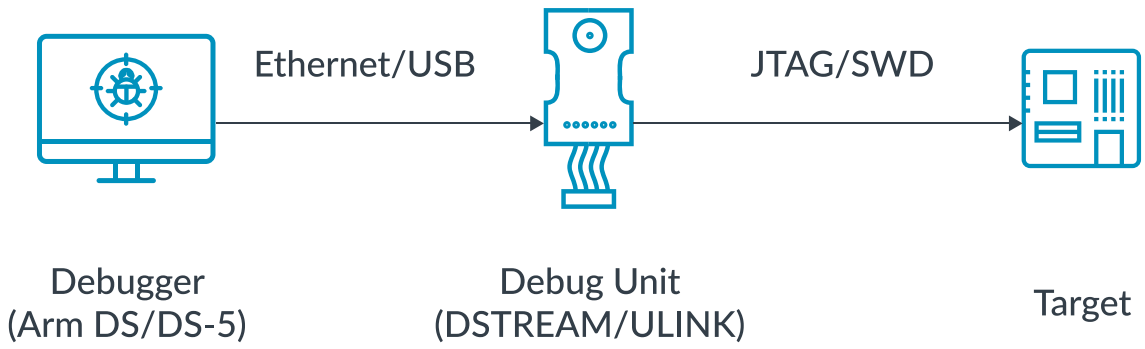
CoreSight-400 and CoreSight-600 provide components to build and validate the debug and trace elements of an SoC built with ARM processors. The components are grouped into categories for controlling and accessing components, trace sources, trace links, trace sinks, and timestamp.

To provide effective debug and trace support for an SoC, the debugger needs to know:

- The devices that are present in the SoC (including the type and configuration details of each device, and connection details such as CoreSight base address).
- How these devices relate or connect to each other (their topology).

The below diagram shows a standard Arm Development Studio (Arm DS) debug and trace connection.

Figure 2-1: a standard Arm Development Studio debug and trace connection



3. How is PCE useful?

Arm DS supports some platforms by default. For platforms Arm DS does not support out-of-the-box, you can use the Platform Editor Configuration(PCE) and the Debug and Trace Service Layer (DTSL) to enable debug connection to virtually any platform which uses supported Arm processors.

The PCE in Arm DS can auto-detect most of the debug and trace information on a platform. However, in some scenarios, the debugger fails to detect certain platform features. The information detected by Arm DS might be incomplete or corrupted, or information might be missing because of:

1. Parts of the SoC are powered down.
2. Devices inside the SoC might interfere with detecting topology.

PCE uses different auto-detection mechanisms for systems based on ADIv5 and ADIv6. This document walks you through the auto-detection process of two Arm reference designs to show you how PCE discovers devices and their topology.

- Arm N1SDP platform is an ADIv5-compliant system which uses CoreSight SoC-400.
- Arm Corstone700 system is an ADIv6-compliant system which uses CoreSight SoC-600 for the debug and trace functions.

4. How does PCE detect the information?

In this document, we focus on the PCE detection process with SWD and JTAG targets. To understand more about the debug architecture, see introduction for the DAP and CoreSight component.

With the PCE detection process, the debugger learns about all the components and CoreSight topology of the system.

The general PCE detection process is:

1. PCE communicates with the DAP to check if it complies with the ADI architecture.
2. Enumerates the APs(Access Ports) and other ports under the DP(Debug Port), like external APBCOM and GPIO control.
3. Detect the CoreSight components.
4. With the base knowledge of all the components on the target, PCE attempts to detect connections between the components.

If you want more information about trace and trace components, see the Understanding Trace architecture guide. The diagram below shows the Corstone-700 and N1SDP development board system CoreSight topologies detected by PCE.

Figure 4-1: CoreSight topologies for Corstone-700

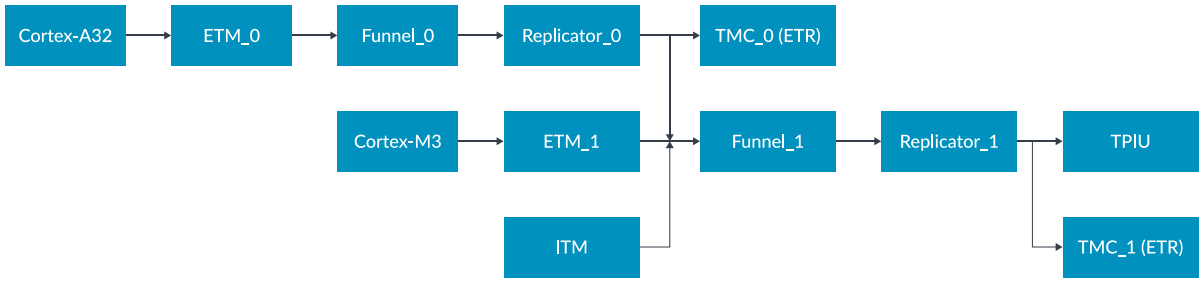
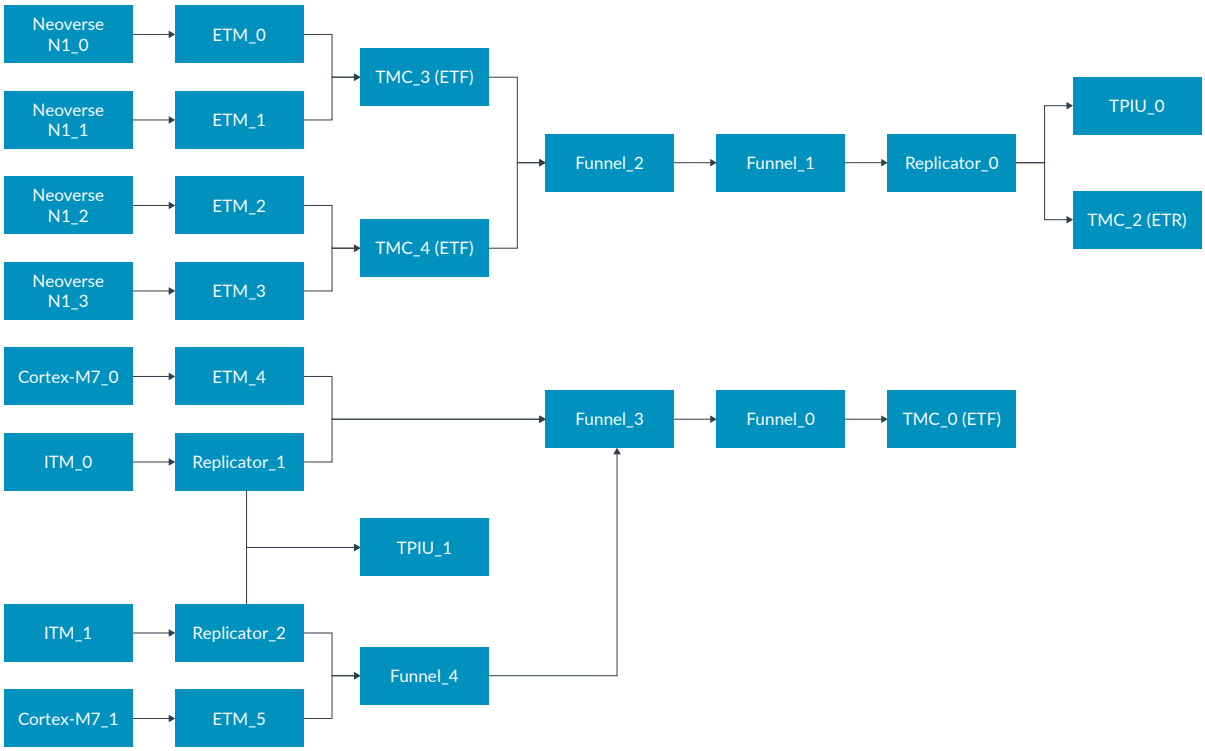


Figure 4-2: CoreSight topologies for N1SDP



The following part of the tutorial introduces the steps in the PCE device and topology detection process using the Arm Corstone-700 and N1SDP development board. It lists the common failure scenarios and the suggested solution in each step. If you are seeing similar situations to what is discussed in this guide and need further assistance, see Requesting help with board bring-up.

Step 1: Get the DR and IR length of the device

To enable communication with all devices, the debugger needs to know the Data Register(DR) and Instruction Register(IR) 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 1s followed by 512 logical 0s. PCE can get the DP number and IR length for each DP. If PCE fails to get the IR length, PCE can't continue any further until this issue is resolved.

When using the PCE, debugger prints the detected CoreSight component information in the PCE Console window. The PCE log provides information about the components found on the device.

On the very start of the PCE log, you can find similar log like below about detect the IR length:

```
Counting devices:
DR Chain [1024]:
11111...11100
Device Count: 2
The total IR length of the scanchain measured as 8
IR Chain [32]:
11111111111111111111100010001
Device 0 detected IR Length = 4
```

```
Device 1 detected IR Length = 4
```

Failure scenario:

1 - Failed to detect scanchain devices

If PCE failed to detect scanchain devices, you can find similiar log like,

```
Device Count: 0
Scanchain detection failure: Failed to detect scanchain devices
Autodetection Failed
```

Solution:

This might caused by several reasons, such as:

- The CoreSight device(s) are not able to go into bypass mode which may related to a low level implementation issue
- The scan chain device(s) are powered down.

It's better to collect a DAP log for the process, please refer to the DAP log collecting KBA How to use the DAP logger tool for detailed steps. Please send the DAP log to Arm support (support@arm.com) for further analysis.

Step 2: Recognize the DAP

PCE reads the 32-bit IDCODE of the device, matching it against a predefined list and identifying the IR length of the component. Typically, the DAP is recognized by PCE as the ARMCS-DP if it's an Arm implemented DAP. You can find the similar log:

```
Reading IDCODEs:
DR Chain [64]:
01101011101000000000010001110111011010111010000000000010001110111
Device 0 has IDCODE = 0x6BA00477 (Manufacturer ID: 0x23B, Part Number: 0xBA00,
 Revision: 0x6)
Device 1 has IDCODE = 0x6BA00477 (Manufacturer ID: 0x23B, Part Number: 0xBA00,
 Revision: 0x6)
Device 0 detected as ARMCS-DP
Device 1 detected as ARMCS-DP
```

After the IDCODE is matched to the DAP of the device, connection with the target is established. Then the tool continues to identify the APs under the DAP.

Failure Scenario:

1 - Locked or secure devices on the scan chain

Some boards have one or more secure or locked devices on the JTAG scan chain. PCE prints the following logs to show information about these devices.

```
Device 0 detected as UNKNOWN_16
```

See Secure devices on my scanchain to see how to program these devices to make the JTAG scan chain accessible.

2 - DAP with a non-standard IDCODE

Some of the in-house design systems introduced a non-standard IDCODE for some season, PCE fails to detect these DAPs. You might find similar log in PCE Console window:

```
Device 0 has IDCODE = 0x1890101D (Manufacturer ID: 0x00E, Part Number: 0x8901,
 Revision: 0x1)
IR Chain [16]:
1111111111110001
Device 0 detected IR Length = 4
Device 0 detected as UNKNOWN_4
```

Please contact Arm support (support@arm.com), the team would help to confirm whether the PCE has the knowledge of the device and provide a solution.

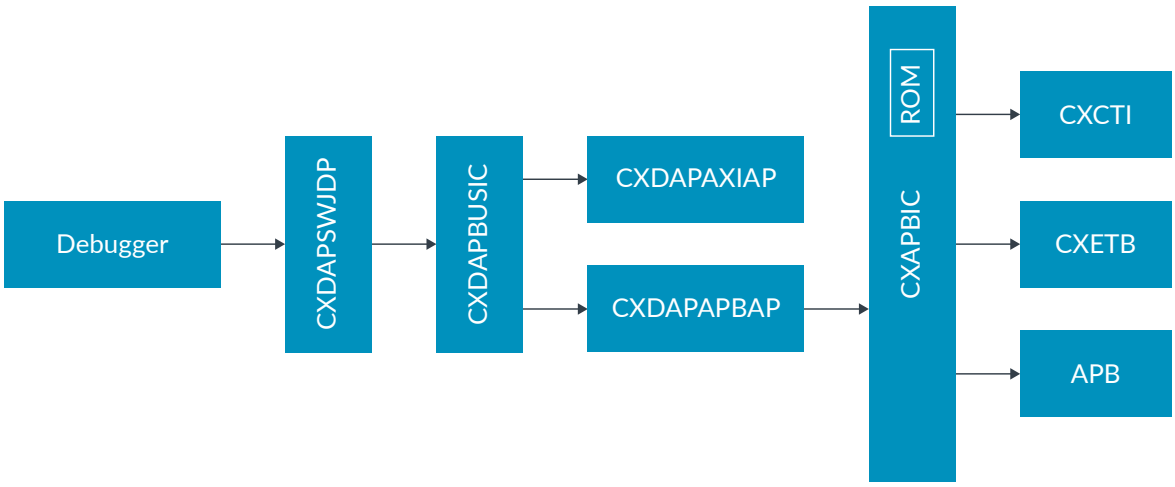
Different auto-detection mechanism for ADIv5 and ADIv6 compliant system

ADIv5 and ADIv6 are different in the ROM table hierarchy design. PCE uses different methods to auto-detect these two kinds of systems. We will descript how PCE deal with these two systems separately.

In systems based on ADIv5, like the CoreSight SoC-400, each MEM-AP has one top-level ROM Table. PCE can get the component location information from the ROM Table entries. There can also be nested ROM Tables under the top-level ROM table.

A system might have a CoreSight topology like the following picture, there are two APs after the DP. And APB-AP has a top-level ROM Table which points to the CoreSight components after this AP.

Figure 4-3: CoreSight topology



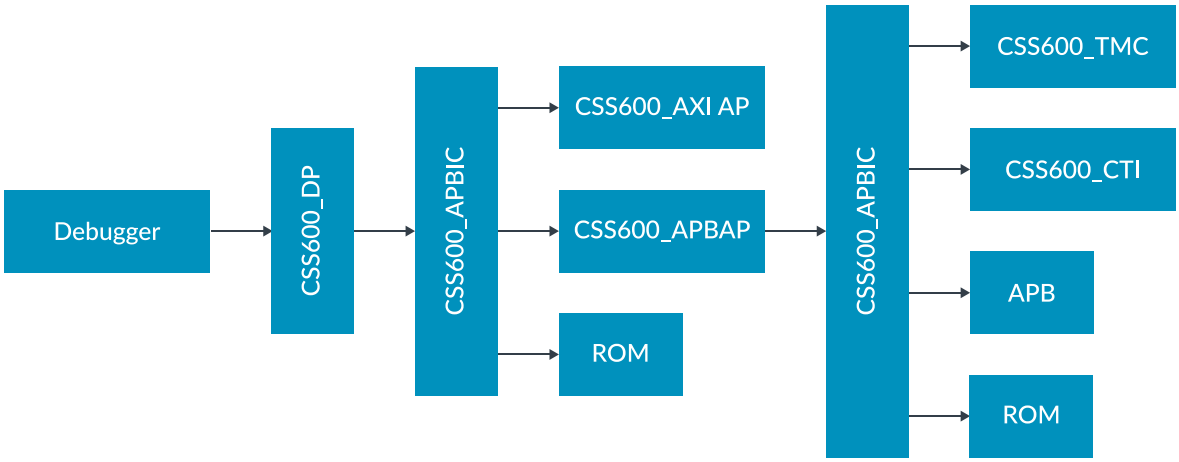
In design of ADIv6-compliant systems, such as Arm CoreSight SoC-600, DP contains a base pointer address which points to the first component on the list of components to be identified, it can be:

- The base address of a ROM Table.
- The address of a debug component, which is the only debug component that is directly accessible from this DP.

See ARM Debug Interface Architecture Specification ADIv6.0 for more detailed information.

For example, in the following SoC-600 system, the DP base pointer register points to a top-level ROM Table which contains the address for the APs and addresses of the debug components that can be directly accessed from this DP. Each MEM-AP has a ROM Table, which contains the address of the debug components behind this MEM-AP.

Figure 4-4: SoC-600 system



Step 3: Power up the DAP

In a correctly designed system, the DAP is in the always on power domain. To request a powerup, the CSYSPWRUPREQ (system powerup) and CDBGPWRUPREQ (debug powerup) signals must be sent. To request the powerup signals, the debugger writes to bits 30 and 28 in the Control/Status Register (CTRL/STAT). The debugger must wait for an acknowledge signal for both powerup requests CSYSPWRUPACK (bit 31) system powerup acknowledge and CDBGPWRUPACK (bit 29) debug powerup acknowledge. Both acknowledge signals are read from the Control/Status Register. More information on the DAP power control can be found in the Understanding CoreSight DAP tutorial.

ADIv5 compliant system

The below steps show the PCE process for an ADIv5-compliant system.

Step 4: Enumerate the APs

Once the IDCODE is matched to the DAP of the device, connection with the DAP is established. The tool then continues with identifying the Access Ports (AP) under that DAP. For example, the N1SDP system has two DAPs, PCE detects the APs after each DAP.

From the PCE console, you can find similar log like:

```
Enumerating AP devices for DAP at scanchain index 0:
Number of AP buses detected: 2
AP types:
APB-AP
AXI-AP
--- --- ---
Enumerating AP devices for DAP at scanchain index 1:
Number of AP buses detected: 2
AP types:
AHB-AP
AHB-AP
```

Step 5: Identify the components after each (MEM-)AP

Each MEM-AP would have a BASE register, it provides the base address of the ROM table. PCE would read the Base register to get the location of ROM table and read the component ID of ROM table to identify whether it's a valid ROM table.

You can find the following log:

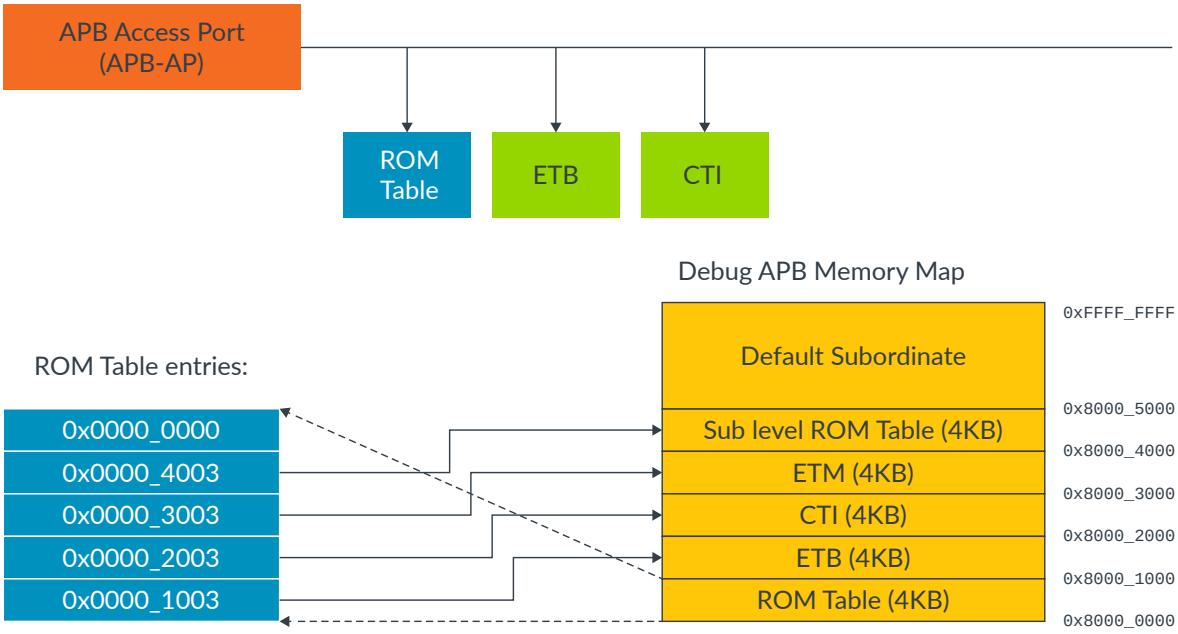
```
Reading ROM table for AP index 0, base address = 0x80000003
Component ID registers:
CID 0: 0x0d
CID 1: 0x10
CID 2: 0x05
CID 3: 0xb1
ROM table part number = 0x7B0
ROM table JEP ID = 0x3B
ROM table JEP Continuation Code = 0x4
ROM table revision number = 0x0
JEDEC used = True
RevAnd = 0x0
Customer modified = 0x0
```

If a valid ROM table is found, PCE would read the entries in the ROM table which points to the memory space of a debug component. ROM table entries can also point to the nested ROM tables, which is referred to as a lower-level ROM Table.

There are two types of ROM Tables, class `0x1` and Class `0x9`. PCE can detect the ROM Tables according to the Peripheral ID registers and Component ID registers.

A typical ROM Table design would be like the following diagram.

Figure 4-5: typical ROM Table design



For detailed information about the ROM table, please refer to ARM Debug Interface Architecture Specification ADIv5.0 to ADIv5.2.

You can find log like:

```
Valid ROM table entries are:
0x00010003 (component base address: 0x80010000)
0x00020003 (component base address: 0x80020000)
0x00090003 (component base address: 0x80090000)
0x000A0003 (component base address: 0x800A0000)
0x000B0003 (component base address: 0x800B0000)
0x000C0003 (component base address: 0x800C0000)
```

Possible Scenario:

1 - Entry present bit is not set.

"`0xXXXXX002` (component base address: `0xXXXXXXXX` - Entry Present bit not set, no device interrogation will occur.)"

This means PCE encountered a ROM table entry which does not have the PRESENT bit set. Such message should be considered with the system design, some components' PRESENT bit might not be set intentionally. The PRESENT bit not being set is only a problem if there is meant to be a debuggable component at that base address and the PRESENT bit is not set.

If an entry is marked as not present, this entry will be skipped by PCE.

PCE will stop scan the ROM table until the entry has a value of `0x00000000`, it means the end of the ROM table.

If you are seeing this message for a ROM table entry for a component that is present in the design, refer to the tutorial about what to do when the ROM table is incorrect or incomplete.

Step 6: Scan the ROM tables

After going through the ROM table entries, PCE would go to the memory space pointed by the entries. PCE would recognize the components by Component and Peripheral ID Registers, which must be present in the register space of every debug component that complies with the ARM Peripheral Identification specification.

Arm debugger has a database of the Arm devices and some customer's designs, if there're some in-house design devices which are not in the components database, you can add the devices into the database manually, see the Common reasons why components and component connections do not appear KBA about how to add components manually, or contact Arm support for help with the component information.

You can find the log as below, if the device's CID or PID is not in the database, PCE would report it as unknown device.

```
Reading peripheral and component ID registers of device at address 0x80010000
Component ID registers:
CID 0: 0x0d
CID 1: 0x90
CID 2: 0x05
CID 3: 0xb1
Peripheral ID registers:
PID 4: 0x04
PID 5: 0x00
PID 6: 0x00
PID 7: 0x00
PID 0: 0x61
PID 1: 0xb9
PID 2: 0x1b
PID 3: 0x00
Peripheral ID = 0x961, JEP-106 code, including continuation = 0x43b, DEVTYPE = 0x32,
 DEVARCH = 0x0, Revision = 0x1
CSTMC found at address 0x80010000
```

Possible Scenarios:

1 - No ROM table is present on this AP

The following log might also be found in the PCE log. This means that PCE could not find a ROM table present for the AP it interrogated. This message usually appears when PCE is interrogating an AP that traditionally does not include a ROM table (like AHB-AP on a Cortex-A/R class system, or AXI-AP). This message is only a problem if there is supposed to be one ROM Table.

```
AXI-AP ROM table base address detected as 0x00000002
No ROM table is present on this AP
```

If this message appears for an AP which does have a ROM table on your board, refer to the tutorial about how to deal with the ROM Table is not found.

This message can also occur if the ROM table base address is wrong and/or set manually. If you believe the ROM table base address might be wrong, refer to the tutorial about ROM Table is incorrect or incomplete.

2 - Unknown device found at address <component base address>, peripheral ID = XXX

PCE recognizes the components by the Peripheral ID and Component ID, PCE takes the component as unknown device if it is not in the Arm DS components database.

According to the component type, it would have different solutions.

- If it's an in-house design component or new-released component, it may not be listed in the database, please refer to KBA Common reasons why components and component connections do not appear about how to add component to the component database.
- Some components aren't detected by Arm DS and it doesn't program up via DTSL, like the HSSTP/SETM. In this case, please program them up yourself.

ADIv6 compliant system

Now we would introduce how PCE detects the ADIv6-compliant system.

Step 3: Read the ROM tables from the DAPs BASEPTR to discover all APs accessed directly though the DAP

In ADIv6 DAP has two Base Pointer Registers BASEPTR0 and BASEPTR1, which provide an initial system address for the first component in the system. Typically, the system address is the address of a top-level ROM Table which indicates where APv2 APs are located.

PCE would access the DP BASEPTR to get the ROM table base address and read the Component ID registers to recognize whether it's a valid ROM table. You can find similar messages in PCE log:

```
Enumerating AP devices for DAP at scanchain index 0:
Identifying component at DAP BASEPTR 0x00000000:
Component ID registers:
CID 0: 0x0d
CID 1: 0x90
CID 2: 0x05
CID 3: 0xb1
Component at DAP BASEPTR is a ROM table
```

PCE would go through the entries in this top-level ROM table. Typically, the entries in the ROM table would point to APs or ROM tables. In this scan, PCE would try to find the APs and any nested ROM table.

If the entries point to other devices, like APBCOM component which provides APB access to a duplex COM Wire Interface link for low-bandwidth communication, the devices would be marked as not a ROM table or AP and be scanned later.

In our example system, there are three entries in this top-level ROM table. The entries point to external APBCOM, external debug ROM and GPIO Control.

PCE completes the first scan for APs and ROM table and finds 4 APs, PCE would print the AP information like:

```
Number of AP buses detected: 4
AP types:
AHB-AP
APB4-AP
AXI-AP
AHB-AP
```

Step 4: Discovering any embedded APs

The detection mechanism different from ADIv5-compliant system is that, the PCE would scan the ROM table in depth to find whether there's nested APs or ROM tables. PCE would go through the entries in the ROM tables after the APs detected in the previous step. It reads the CID and PID to decide whether it's a ROM table or AP, but not compare the ID with the CoreSight component to decide the CoreSight components' type.

If there's any locked device been found, such as the APBCOM, after unlocking the APBCOM, PCE would also rescan the ROM table after the APBCOM to find any embedded APs. The same ROM Table may be detected for twice or three times in the auto-detection process.

Possible failure scenario:

1 - Component is not a ROM table or AP

You may find the following information in the PCE log:

```
Peripheral ID = 0xd22, JEP-106 code, including continuation = 0x43b, DEVTYPE = 0x0,
 DEVARCH = 0x2a04, Revision = 0x0
Component is not a ROM table or AP
```

This means that PCE encountered a component that could not be identified as a ROM table or an AP.

For ADIv6.0 compliant boards, the PCE will firstly read ROM table from the DAPs BASPTER to discover all APs accessed directly through the DAP. Then scan for embedded AP devices. The last scan is read the ROM tables to identify devices.

This message appears during the AP discovery phase when the system is detected, if there's no embedded AP. It's normal message for this case.

If the message appears during the ROM table read phase when debug and trace components are identified, it indicates that the component is unrecognized which could lead to the component not being detected correctly. Also, if this message appears for a component which is a ROM table or an AP, it indicates PCE was unable to recognize the component correctly. In either situation, refer to the tutorial about the ROM Table issue in PCE detection.

Step 5: Auto-detect the devices

After all the APs directly connected with DP and embedded APs been found, PCE would start to detect the CoreSight devices in the system. It would scan the ROM tables and nested ROM tables of each AP to find all the CoreSight devices.

In the PCE detection of Corstone700 system, the third scan is used to identify the CoreSight components. PCE would print log like:

```
Reading peripheral and component ID registers of device at address 0x82040000
Component ID registers:
CID 0: 0x0d
CID 1: 0x90
CID 2: 0x05
CID 3: 0xb1
Peripheral ID registers:
PID 4: 0x04
PID 5: 0x00
PID 6: 0x00
PID 7: 0x00
PID 0: 0x0c
PID 1: 0xbd
PID 2: 0x1b
PID 3: 0x00
Peripheral ID = 0xd0c, JEP-106 code, including continuation = 0x43b, DEVTYPE = 0x13,
 DEVARCH = 0x24a13, Revision = 0x1
CSETM found at address 0x82040000
```

Step 6: acquire the information of the CoreSight components

After all the ROM Tables been scanned, the PCE would start to scan the connections between the devices. It would be the same process for PCE to detect on ADIv5-compliant and ADIv6-compliant systems.

Before testing Affinity registers to get the topology information, PCE would firstly acquire the information of the CoreSight components. For example:

- TPIU - PCE needs to get the TPIU supported port size.
- ETM - PCE needs to get the version of the ETM and whether it can support timestamps, context ID, data address and value trace.
- TMC - PCE needs to get the configuration of each TMC, it's used as ETR, ETF or ETB.

PCE would print log like:

```
Acquiring device info for CSTMC_0 (0x80010000):
TMC Device Config: ETF
Memory Width: 128
Ram Size: 16,384 Bytes
Device Type: Trace Link
Drain Source: ATB
Acquiring device info for CSTPIU_0 (0x80130000):
Supported Port Sizes: 1-32
Acquiring device info for CSETM_0 (0x82040000):
CSETM_0 (0x82040000) is OS Locked
CSETM_0 (0x82040000) OS Lock successfully removed
Device supports timestamps
Device supports context IDs
Device supports cycle accurate trace
Device does not support data address trace
```

```
Device does not support data value trace
Device does not support trace range
ETM Version: 4.2
```

Failure Scenario:

1 - Failed to read registers to identify component: Failed to read 16 bytes from address `0xXXXX0FF0` on CSMEMAP_<AP number>

You may find the error information reports that, PCE failed to read the identify register to

```
Failed to read registers to identify component: Failed to read 16 bytes from address
 0x80090FF0 on CSMEMAP_0)
```

For the CoreSight components, registers on the offset from `0xFF0` to `0xFFC` are the Component ID registers. This might be caused by different reasons that PCE failed to read the Component ID registers.

Check whether the core is in a stable power state, make sure what is running on the target when doing the PCE. For example, if you stop the target at u-boot then u-boot is likely to only be running on 1 core and the other one is powered down.

Please help to collect the DAP log, then send the log information in PCE console and DAP log to Arm support (support@arm.com). Support team might find more information from the log file. Consult the How to use the DAP logger tool for instructions on how to capture a DAP log.

2 - Failed to read x bytes from AP <AP number> @ <component debug register base address>

This means that PCE failed to read from a component's debug registers. Debug register reads usually occur when trying to identify the component during the auto-detection process. A read failing might result in the component not being added to the platform configuration. If the component is missing from the platform configuration, please refer to Common reasons why components and component connections do not appear for this issue. You can also add the component manually according to the Arm Debugger Manual Configuration Tutorial.

3 - Failed to read device info for device <trace component> (<trace component base address>): Failed to read x bytes from address <trace component debug register address> on CSMEMAP_<AP number>

This means that PCE was unable to read the device information for the CoreSight component. This is usually due to the component being inaccessible when the read was performed. See Common reasons why components and component connections do not appear for more details on this behavior.

Step 7: connections between the CoreSight components

Typically, the trace data in the system would go through from trace source, trace link and then arrive at trace sink. PCE would detect the connections that exist in the system design, such like:

- Connections between the cores and ETMs
- Connections between ETMs and CTIs,

- Connections between funnels and TMCs.

Debugger can get the information needed to debug and trace and PCE would generate the platform configuration for the target, including `*.sdf`, `dtsl_config_script.py`, `project_types.xml`.

PCE would print logs:

```
Testing affinity register topology for V8 core clusters:
Core Neoverse N1_0 (0x82010000) master cluster ID is 0
MASTER = Neoverse N1_0 (0x82010000)              SLAVE = CSCTI_5 (0x82020000)
 Trigger = 1
MASTER = Neoverse N1_0 (0x82010000)              SLAVE = CSETM_0 (0x82040000)
MASTER = CSETM_0 (0x82040000)          SLAVE = CSCTI_5 (0x82020000)  Trigger = 4
```

In some scenarios, PCE might not find some of the connections. This can be solved by:

- Determining which connections are missing.
- Checking the connections with the hardware design or the documentation.
- If the missing connection do exist in the design, you can find the tutorial about how to add the links manually.

PCE generates the platform configuration with the detected information, there might be different cases according to the detection result.

1. PCE gets enough component connection information to generate a platform configuration that can be used to debug-only,
2. PCE gets enough component connection information to generate a platform configuration that can be used to debug and trace,
3. PCE doesn't get enough information to debug the target (i.e. missing cores, CTI, and trace component connections).

In case 1), the CTI component connections are found, but maybe not all of the trace component connections are found. In this case, PCE can generate a debug only platform configuration and the user can still debug the target.

In case 2), PCE thinks the platform configuration is complete, but it is still worth the user going through the SDF file device list and component connections to make sure all the information is correct and no items are missing. PCE can generate a debug and trace platform configuration.

In case 3), PCE is missing critical information and the user needs to manually configure some, if not all, of the platform configuration. PCE can't generate platform configuration without adding the critical information.

You may find the following reminder in the log:

```
Platform "Arm - Test Platform" did not build successfully: Platform contains
 incomplete or invalid topology, or other warnings, and cannot be built
 automatically. Build this platform manually to access the build options.
```

In some cases, you might modify the .sdf file manually, after save the modification please rebuild the platform configuration by right click the .sdf file and choose the Build Platform.

Failure scenario:

1 - Failed to get ATB master topology for device <trace component> (<trace component base address>): Failed to read x bytes from address <trace component debug register address> on CSMEMAP_<AP number>

This means that PCE was unable to determine how a trace master component is connected to the rest of the CoreSight infrastructure. PCE usually tries to determine the connection multiple times to test for a different result. You need to add the appropriate connection(s) for the component in the Component Connections tab of the platform configuration's .sdf file to make the trace component available for tracing purposes. Refer to the tutorial for information on how to add the necessary connection(s).

2 - Failed to get core trace topology for device <core> (<core base address>): Failed to write x bytes to address <core debug register address> on CSMEMAP_<AP number>

This means PCE was unable to determine the connection(s) between the core and the trace subsystem. PCE usually tries to determine the connection multiple times to test for a different result. You might also see the below messages several times for the same core:

```
Integration test postamble for device <core> (<core base address>) failed: Failed to
 write x bytes to address <core debug register address> on CSMEMAP_<AP number>
Failed to get trigger topology for device <core> (<core base address>): Failed to
 write x bytes to address <core debug register address> on CSMEMAP_<AP number>
Failed to read affinity registers for device <core> (<core base address>): Failed to
 write x bytes to address <core debug register address> on CSMEMAP_<AP number>
Failed to get MPIDR cluster topology <core> (<core base address>): Failed to write x
 bytes to address <core debug register address> on CSMEMAP_<AP number>
```

You need to add the appropriate connection(s) for the core in the Component Connections tab of the platform configuration's .sdf file to make the core available for synchronous debug control and tracing purposes. Refer to the tutorial about how to add the necessary connection(s).

3 - Failed to get core trace topology for device <trace component> (<trace component base address>): Failed to read x bytes from address <trace component debug register address< on CSMEMAP_<AP number>

This means PCE was unable to determine the connection(s) between the trace component and the trace subsystem. PCE usually tries to determine the connection multiple times to test for a different result. You might also see the below messages several times for the same core:

```
Integration test postamble for device <trace component> (<trace component base
 address>) failed: Failed to write x bytes to address <trace component debug
 register address> on CSMEMAP_<AP number>
Failed to get trigger topology for device <trace component> (<trace component base
 address>): Failed to write x bytes to address <trace component debug register
 address> on CSMEMAP_<AP number>
Failed to read affinity registers for device <trace component> (<trace component
 base address>): Failed to write x bytes to address <trace component debug register
 address> on CSMEMAP_<AP number>
```

You need to add the appropriate connection(s) for the component in the Component Connections tab of the platform configuration's .sdf file to make the trace component available for tracing purposes. Refer to the tutorial about how to add the necessary connection(s).

If you are uncertain about the responsiveness or presence of certain components in the board design, you can interact with these components on a low level using the CoreSight Access Tool (CSAT) or the CoreSight Access Tool for SoC600 (CSAT600). Below is reference material for CSAT and CSAT600:

- CoreSight Access Tool (CSAT) User Guide
  - The user guide for the CSAT tool.
- CoreSight Access Tool for SoC600 (CSAT600) User Guide
  - The user guide for the CSAT600 tool.
- Low Level Debug using CSAT on Armv8-based Platforms
  - A tutorial which gives an overview of performing low level debug using the CoreSight Access Tool (CSAT) with an Armv8 target.

The tutorial references DS-5 exclusively, but the content still applies to Arm Development Studio.

- Low Level Debug using CSAT on Armv7-based Platforms
  - A tutorial which provides information about performing some basic debug operations on an Armv7-based platform using the CoreSight Access Tool (CSAT).

The tutorial references DS-5 exclusively, but the content still applies to Arm Development Studio.

If you are working a CoreSight-enabled target, you can also obtain a log of the low level activity between a DSTREAM family unit (DSTREAM, DSTREAM-ST, DSTREAM-PT, and DSTREAM-HT) and the CoreSight components using the DAP logger tool. Consult the How to use the DAP logger tool for instructions on how to capture a DAP log.

If you are debugging with a ULINKpro, you can also get the log of the low level activity between the ULINKpro and the CoreSight components following the instructions on How can I use the ULINK debug probe to collect the debug log in Arm DS?

Note: You will need a good understanding or knowledge of the CoreSight Architecture, the CoreSight components, the board's CoreSight topology details, and processor-specific debug information (such as CoreSight register offsets) in order to interpret the DAP logger tool output.

5. Related information

The following resources are related to this guide:

- Arm Debugger Manual Configuration Tutorial
- Common reasons why components and component connections do not appear
- Common .sdf errors and warnings and what can be done to solve them
- Help with connecting to new targets
- Help with debugging and tracing targets
- How can I use the ULINK debug probe to collect the debug log in Arm DS?
- How to use the DAP logger tool
- Secure devices on my scanchain
- Understanding the CoreSight DAP
- What is lock device?