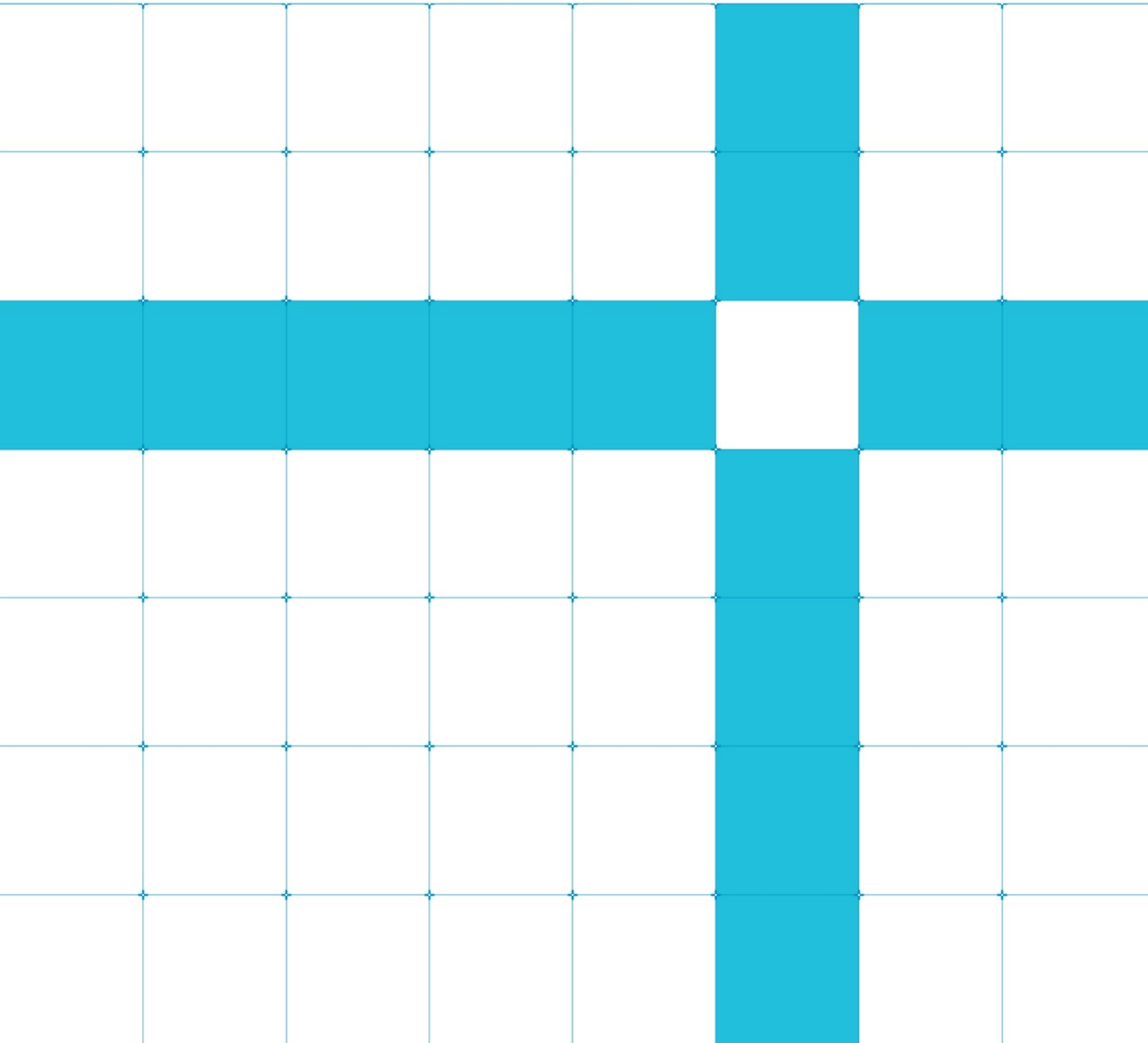
Arm Debugger Guide

Understanding the CoreSight DAP

Version 0.0



Arm Debugger Tutorial

Understanding the CoreSight DAP

Release Information

Document History

Version	Date	Confidentiality	Change
0.0	17 06 2020	Non-Confidential	First version

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

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.

Web Address

.**http://www.arm.com**.

Contents

1 Overview

The aim of this guide is for you to gain a better understanding of:

- What a CoreSight Debug Access Port (DAP) is and how it works.
- What an Access Port (AP) is and how it works.
- What a ROM Table is and how it is implemented.
- Common DAP-related issues that you might encounter when auto-detecting a board with the Arm Development Studio Platform Configuration Editor (PCE).

This tutorial focuses on these topics from an Arm Development Studio perspective. It is also useful for readers interested in CoreSight DAP knowledge.

2 What is a Debug Access Port?

Typically, CoreSight devices are behind a CoreSight Debug Access Port (DAP). Arm CoreSight technology is used to debug and trace complex SoC designs. A DAP is a Debug Port (DP) that is connected to one or more Access Ports (APs). A DP provides a connection from outside the SoC to one or more APs. Usually, the connection is based on a simple physical interface like JTAG or Serial Wire (SW).

An AP provides a connection from the DP to a subsystem on the SoC. Many subsystems consist of multiple debug components that are arranged in a memory map. An AP provides the connection to these memory mapped components. If more than one subsystem is accessed, more than one AP is used.

DAP implementations follow one of these Arm Debug Interface (ADI) Architecture Specifications:

- Arm Debug Interface Architecture Specification ADIv5.0 to ADIv5.2
- Arm Debug Interface Architecture Specification ADIv6.0

These architecture specifications describe how debug tools, like Arm Development Studio, interact with CoreSight devices.

CoreSight SoC-400 implements ADIv5.x. CoreSight SoC-600 implements ADIv6.

In the Arm Development Studio platform configurations, the DP is represented by a <name of IP creator>CS-DP device. For example, an Arm-implemented DP is an ARMCS-DP.

2.1. DP versions

The ADI and CoreSight SoC versions implemented determine the DP version that is used.

ADIv5.x defines DPv0, DPv1, and DPv2.

ADIv5.x implementations provide an external debugger physical connection interface to debug or trace a SoC. The possible physical connection interfaces are:

Physical connection interfaces	JTAG interface	Serial Wire Debug (SWD) interface
JTAG Debug Port (JTAG-DP)	X	
Serial Wire Debug Port (SW-DP)		X
A combined Serial Wire/JTAG Debug Port (SWJ - DP)	X	X

To allow the most physical connection flexibility, most ADIv5.x SoC implementations use a SWJ - DP.

ADIv6.0 defines DPv3, which expands on the features and capabilities of the previous DP versions. ADIv6 introduces a layering system that provides memory-mapped access to all parts of a system from multiple different agents, including external debuggers and on-chip software. These layers include:

- The physical layer:

    - The physical pins on a target to connect a debugger to a target
    - Examples of physical layers include:
        - JTAG connector
        - SWD connector
        - USB connector
        - PCIe connector
        - IP sockets
    - The type(s) of physical connections that a SoC uses are referred to as debug links.
- The protocol layer:
    - The JTAG and SWD state machines
- The link layer:
    - The mechanism to perform basic accesses to DP(s) registers, and AP(s)
- The AP layer:
    - Provides access to subsystems
    - Read What is an Access Port? for more information on APs

The following diagram is an example of an ADIv6 system:

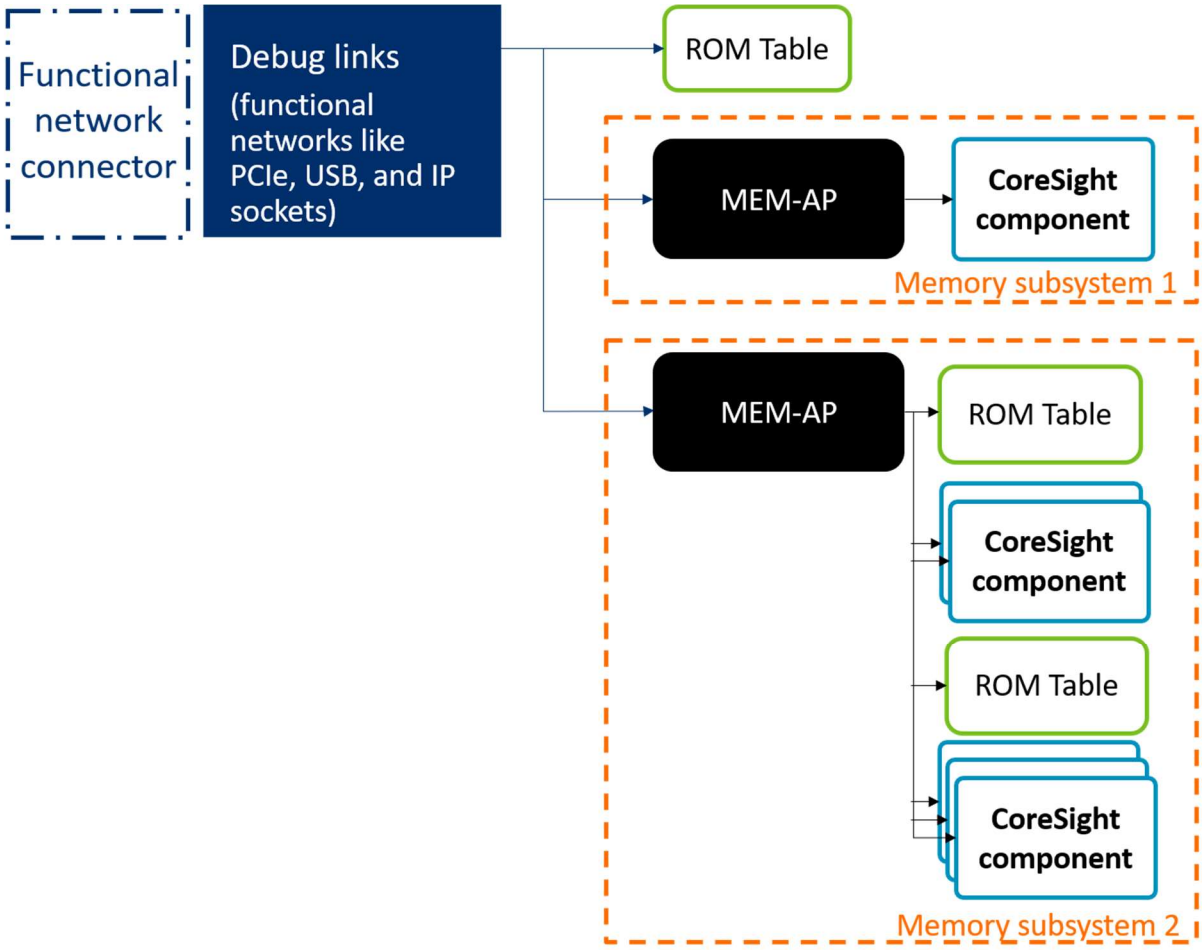


Figure 1 ADIv6 example system

The following diagram shows an ADIv5.x or ADIv6 external debugger connection:

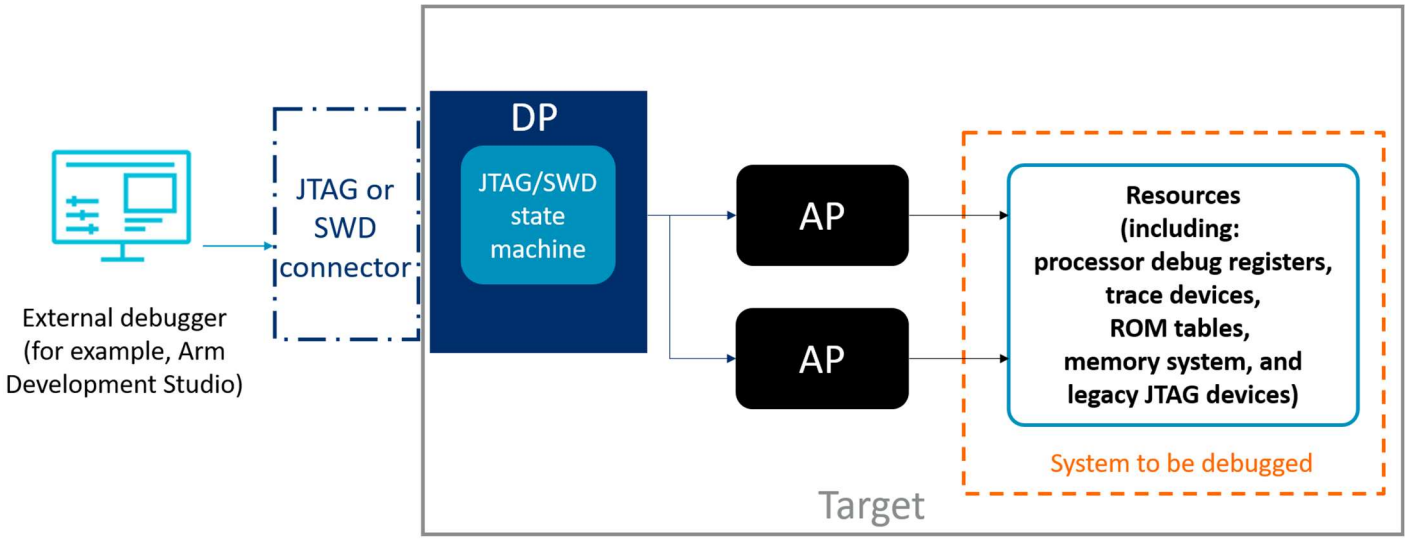


Figure 2 External debugger connection with ADIv5.x and ADIv6

2.2. DAP power control model

The SoC designer determines which power domain the CoreSight devices are in, and how the devices are powered. Typically, a SoC divides the debug devices into a separate debug power domain, and other devices into system power domains. Usually, the SoC power controller determines when and which devices are powered up.

ADIv5.x and ADIv6 define two pairs of power control signals in the DP CTRL/STAT register:

- CDBGPWRUPREQ and CDBGPWRUPACK
    - CDBGPWRUPREQ is a signal from the debug interface to the power controller to fully power the system and ensure that clocks are available to the debug power domain. This ensures that the debugger can access enough debug resources of the CoreSight devices to determine their state. This signal also allows the debugger to perform debug operations like run or step.
    - CDBGPWRUPACK is a signal from the power controller to the debug interface to acknowledge the CDBGPWRUPREQ signal.
- CSYSPWRUPREQ and CSYSPWRUPACK
    - CSYSPWRUPREQ is a signal from the debug interface to the power controller to fully power the system and ensure that clocks are available to the system power domain. This ensures that the debugger can access the non-CoreSight components of a SoC, like main memory and interconnects.
    - CSYSPWRUPACK is a signal from the power controller to the debug interface to acknowledge the CSYSPWRUPREQ signal.

These signals are requests to the system power and clock controller to enable external debugging. The system power and clock controller should honor these requests.

In ADIv6, because CDBGPWRUPREQ and CSYSPWRUPREQ are pieces of DP functionality that are not directly accessible to functional networks like PCIe, the Granular Power Requester (GPR) is the primary powerup request mechanism. The GPR is in the ROM Table(s), or in ADIv5-compliant systems, in the standalone components.

A ROM Table points to debug components. A GPR in a ROM Table uses the debug components pointed to by the ROM Table to allow a debugger to detect the power domain of the components. The debugger requests power from the power controller only for the domains that are required for status checking and debug operation purposes.

The following diagram shows a ROM Table with a GPR, where the GPR requests power to two different power domains:

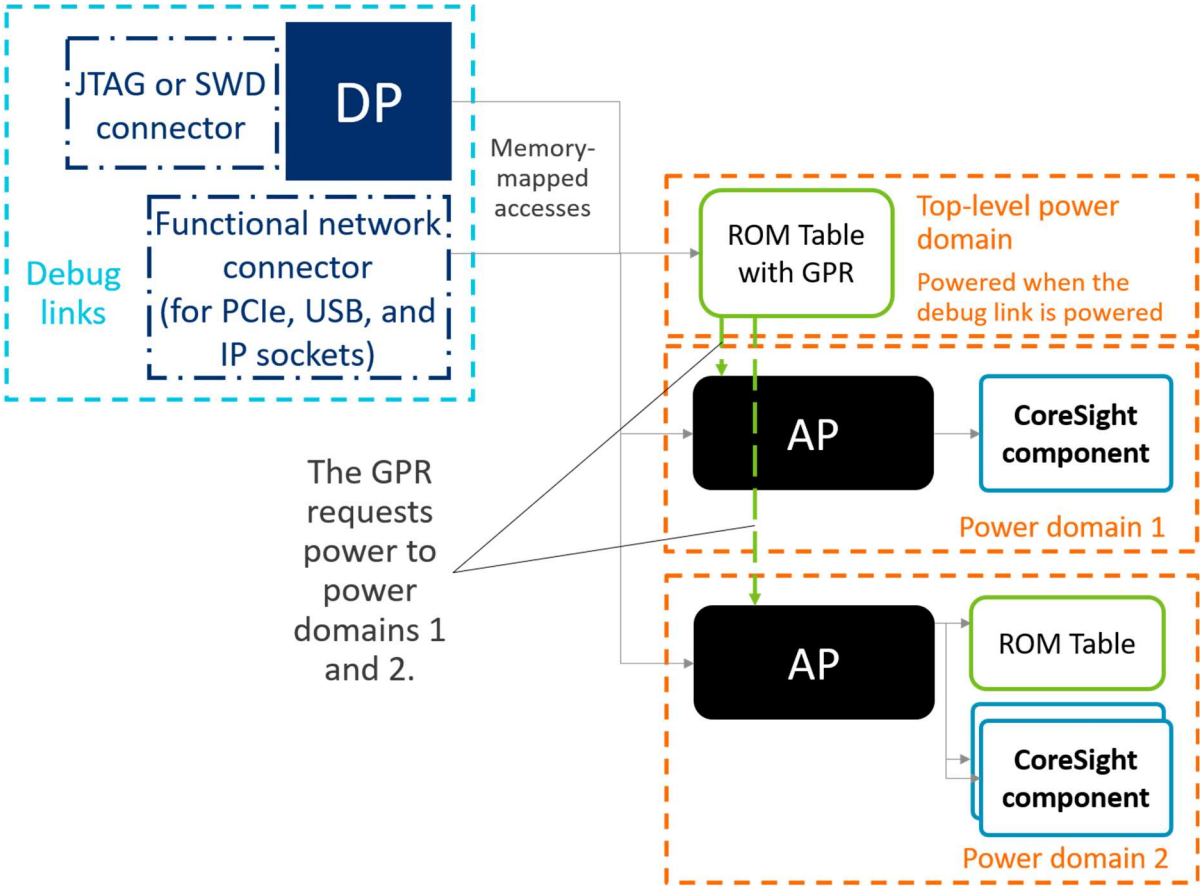


Figure 3 Example power domains with a top-level power domain at the first level

The debugger might need to connect to components that are not pointed to by a ROM Table to debug a SoC. For example, a debugger might need to connect to the system interconnect configuration components. The GPR can support issuing powerup requests to further power domains. These further power domains are referred to as system power domains. The GPR enables a debugger to request power up to components in system power domains across the SoC.

What is ROM Table? provides more information on ROM Tables.

3 What is an Access Port?

An Access Port (AP) is a port that is connected to a DP or debug link. An AP provides a bridge into another system on the SoC.

A Memory Access Port (MEM-AP) provides a window into a memory system. This window allows memory-mapped accesses to debug resources. Examples of debug resources are:

- Debug registers of a core processor
- Debug registers for trace components such as Embedded Trace Macrocell (ETM) and Trace Memory Controllers (TMC) instances, for example, Embedded Trace FIFO (ETF) and Embedded Trace Router (ETR)
- Debug registers for CoreSight links like Cross Triggering Interfaces (CTIs)
- ROM Tables
- Memory systems

ADIv5.x defines APv1. ADIv6 defines APv2. APv2 is not backwards-compatible with APv1.

Here are the AP types that are available:

- Advanced Peripheral Bus Access Port (APB-AP)
    - For interfacing to APB memory systems
    - Typically, CoreSight component debug registers, like for Cortex-A and Cortex-R processors, are accessible through this AP.
- Advanced High-performance Bus Access Port (AHB-AP)
    - For interfacing to AHB memory systems
    - Typically, Cortex-M class debug and trace registers are accessible through this AP.
- Advanced eXtensible Interface Access Port (AXI-AP)
    - For interfacing to AXI memory systems
- JTAG Access Port (JTAG-AP)
    - For interfacing to legacy components like pre-CoreSight processors. For example, Arm7, Arm9, and Arm11 processors.

The following diagram demonstrates the different AP types in a system with a JTAG/SWD debug link. An ADIv5-compliant system is used in the diagram:

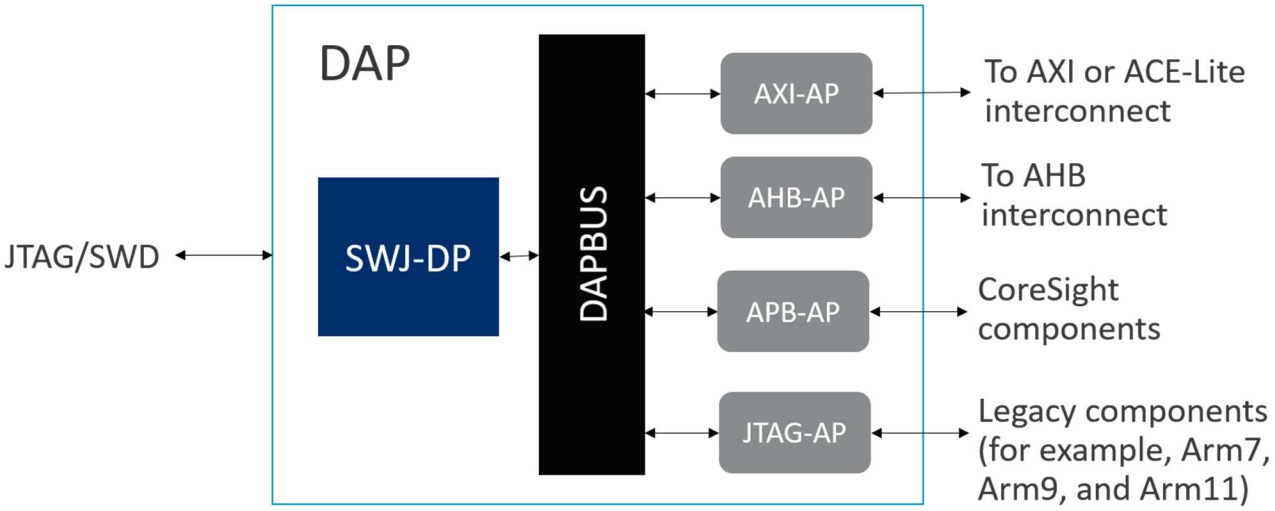


Figure 4 DAP diagram

In ADIv5, a DP supports up to 256 APs. In ADIv6, only the address space limits how many APs a DP can support.

The Identification Register (IDR) of the AP identifies the AP designer, variant, and type. If the IDR value is zero, this indicates that the AP is not present in the system. The IDR is defined in the ADI architecture specification.

In ADIv6, on-chip software is permitted to access the AP layer, which enables on-chip debug software to access multiple systems using the same APIs as external debuggers.

4 What is a ROM Table?

Each ROM Table on the SoC contains a listing of the components that are connected to the DP or MEM-AP. These listings allow an external debugger or on-chip software to discover the CoreSight devices on the SoC. Systems with more than one debug component must include at least one ROM Table. ROM Tables are connected either to DPs or MEM-APs.

A ROM Table entry either contains an address offset for a component on the SoC, or a pointer to another ROM Table. You calculate the base address of the component by adding the component address offset to the ROM Table base address. If the ROM Table entry is a component address offset, the PRESENT bit of the ROM Table entry indicates whether the component is present in the system. The end of a ROM Table is marked by an all 0x0 entry or an entry at ROM Table offset `0xEFC`.

In both ADIv5.x and ADIv6, ROM Tables can be nested, with no limit on the depth of the nesting. Nesting in this context means one ROM Table can point to another ROM Table.

The following diagram shows the placement and possible entries of a ROM Table that is connected to an APB-AP:

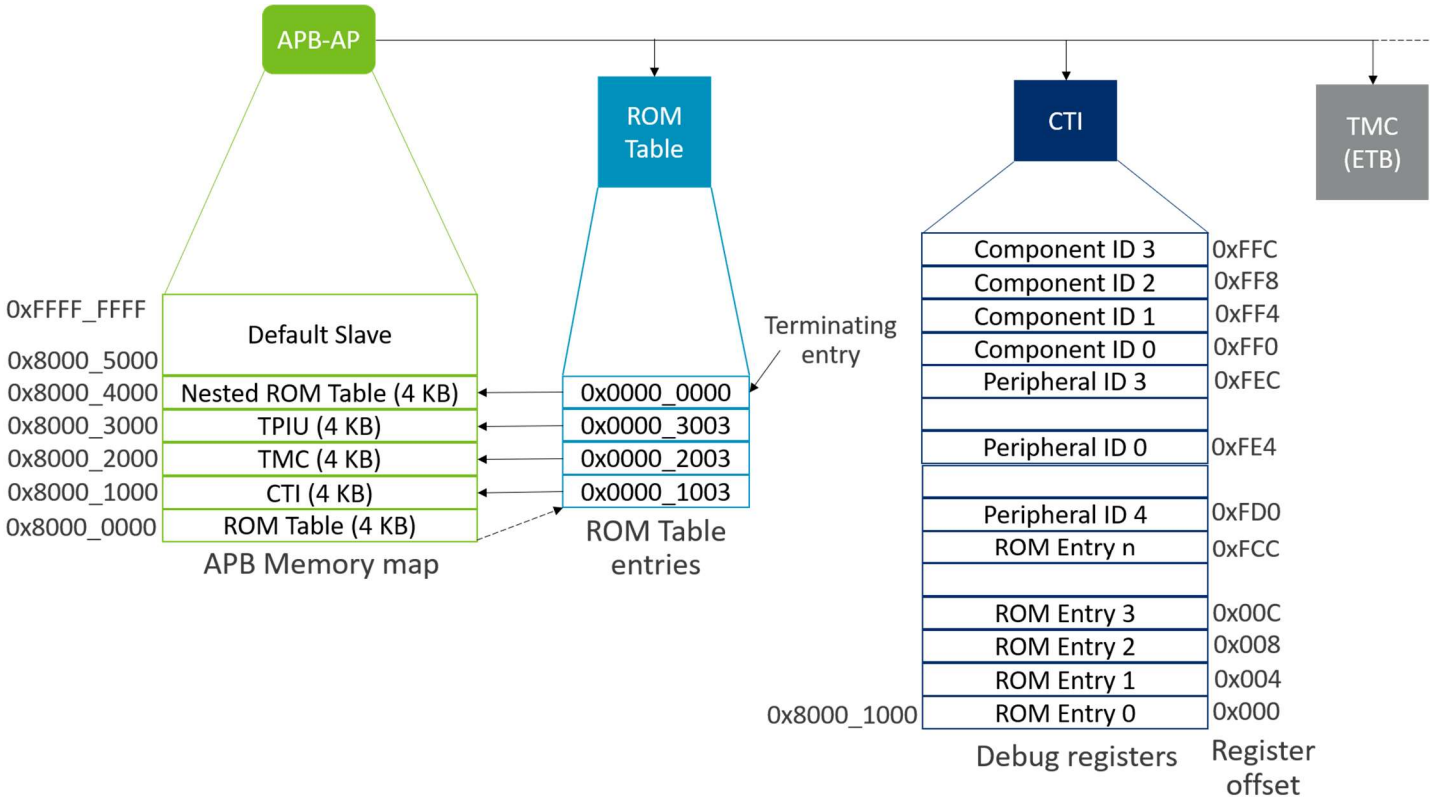


Figure 5 ROM Table diagram

The location of the top-level ROM Table is determined by one of the following:

- The ROM Table base address register of the MEM-AP, BASE.
- On ADIv6 targets, by the ROM Table base address registers of the DP, BASEPTR0-BASEPTR1.
- OS or debug monitor knowledge of the system memory map.

5 Common DAP-related autodetection issues

This section covers some of the common DAP-related issues that Arm Development Studio users might encounter when auto-detecting a board with the Platform Configuration Editor (PCE).

5.1. The DAP is powered down

In systems where a DAP is present, the PCE autodetection process communicates with the system DAP to determine which CoreSight components are on the board. To communicate with the DAP, the debugger requests that the DAP is powered up either by asserting the CDBGPWRUPREQ signal or by communicating with the GPR.

If the DAP has not been powered up, a Failed to power up DAP error message is shown in the PCE Console, and the autodetection process for the DAP stops. PCE determines whether the DAP has been powered up using one of the following methods:

- PCE can communicate with the board through the DAP.
- A powerup acknowledgment is received through the CDBGPWRUPACK signal.
- PCE can communicate with the board after the GPR powers up the necessary power domains.

If PCE autodetection stops because the DAP is not powered up

1. If you are using a CoreSight SoC-600 or an ADIv6-compliant board, check that the debugger and debug probe supports these targets by consulting your debugger and debug probe reference material.
    - The latest versions of Arm Development Studio support CoreSight SoC-600 or ADIv6-compliant boards.
    - Debug probe firmware installed or upgraded using Arm Development Studio support CoreSight SoC-600 or ADIv6-compliant boards.
    - DS-5 does not support CoreSight SoC-600 or ADIv6-compliant boards.
2. Ensure that the power domain containing the DAP is powered up on the board. Consult the board designer, manufacturer, or documentation to determine if there are any items that can prevent DAP powerup.
    - Some boards might require more steps to power up the DAP, for example:
        - Physically configuring the board, like setting DIP switches or making certain board connections
        - Having a debug-capable image on the board
        - Configuring the board power controller
3. If the board design is still configurable, for example, the board is an FPGA, and the DAP is not powering up due to the hardware configuration, have your hardware designer modify the board design so the DAP powers up when a debugger powerup request occurs.

You can use the **CoreSight Access Tool for SoC600 (CSAT600)** or **CoreSight Access Tool (CSAT)** tool to verify that the DAP is powering up.

5.2. Wrong number or type of APs found

As part of the PCE autodetection process, PCE tries to detect which APs are present behind the DAP.

If an AP is marked as not being present in the system because its IDR value is 0x0, PCE skips interrogating the components that are connected to the AP. This means that the components connected to the AP are not added to the platform configuration.

If APs are not terminated according to the board integration documentation, PCE cannot determine the number and the type of APs in the system. If the number of APs present cannot be determined, PCE might assume that the maximum number of possible APs are present. If PCE makes such an assumption, after the autodetection process completes, a large number of APs are shown in the PCE Console view.

If PCE autodetection does not find an AP or the wrong number or type of AP is found

Consult with the board designer, manufacturer, or documentation to determine which APs are present behind the DAP. Compare the AP number and types that are included in the board resources to the APs found by the PCE autodetection process. The results of the PCE autodetection process are shown in the PCE Console view.

- If an AP is not found during the autodetection process:
    a. Check whether the IDR value for the AP is 0x0.
        i. If supported, the AP IDR value is accessible using the memory mapped AP interface. Consult your board documentation for where and how this information is accessible.
        ii. If the IDR value is 0x0, correct the IDR value in the hardware design if possible. If correcting the IDR value is not possible, follow the instructions at the end of this section to manually configure a target with PCE and add the missing AP.
    b. If the IDR value is not 0x0, follow the instructions at the end of this section to manually configure a target with PCE.
- If the wrong type of AP is found:
    - Check that the AP Type (AP_TYPE) for the AP is correct. The AP Type value is found by clicking the AP under Devices in the platform configuration SDF file. If the AP Type value is incorrect, correct the TYPE value in the AP IDR in the hardware design if possible. If correcting the IDR TYPE value is not possible, follow the instructions at the end of this section to manually configure a target with PCE and change the AP Type value in the platform configuration.
- If more APs are found than expected:
    - Run the CoreSight integration tests found in the processor's Integration Manual.

Instructions on how to manually configure a target with PCE:

- Consult the **Manual platform configuration section of the Arm® Development Studio User Guide.**
- Consult the **Arm Debugger Manual Configuration Tutorial.**

- Watch the following videos:
  PCE Manual Platform Configuration (1 of 4)
  PCE Manual Platform Configuration (2 of 4)
  PCE Manual Platform Configuration (3 of 4)
  PCE Manual Platform Configuration (4 of 4)
  Note: These videos focus on DS-5, but most of the content remains the same for Arm Development Studio. DS-5 is a retired product.

5.3. The ROM Table is not found

As part of the autodetection process, PCE interrogates all the ROM Tables on the board to determine which components are present. In the PCE Console view, if PCE cannot find a ROM Table for an AP, a No ROM Table is present on this AP message is shown. If the ROM Table location information is wrong, PCE does not find the ROM Table and none of the components listed by the ROM Table are added to the platform configuration.

If a ROM Table is not found

In the PCE Console view, look for No ROM Table is present on this AP messages for APs which do have a ROM Table, or DPs and APs that are missing all their associated components. This helps you identify which ROM Table(s) were not found. To correct a platform configuration with missing ROM Table(s):

1. Check the ROM Table base address, or whether the OS or the debug monitor knowledge of the system is correct.

   The ROM Table base address is listed in:

   - The device information in the platform configuration SDF file. To open this information in the SDF file, go to Devices and click the device.
   - The memory mapped AP MEM-AP BASE register.
   - If using an ADIv6-compliant target, the DP ROM table base address registers, BASEPTR0-BASEPTR1.

   The OS or debug monitor knowledge of the ROM Table(s) location depends on the OS or debug monitor that is used. Consult your OS or debug monitor documentation.

   If the ROM Table location information is incorrect, correct the information in the hardware design or the software if possible. If a correction is not possible, manually set the ROM Table information by:

   a. Under Devices in the platform configuration SDF file, click the item missing the ROM Table such as a DP or an AP.
   b. Manually set the ROM Table location information.
   c. Right-click on the item missing the ROM Table and select Read ROM Tables. This starts the autodetection process again with the corrected ROM Table location information.

2. If the ROM Table location information is correct, follow the instructions for manually configuring a target with PCE and add the missing ROM Table components:

   - Consult the **Manual platform configuration section of the Arm® Development Studio User Guide.**
   - Consult the **Arm Debugger Manual Configuration Tutorial.**

- Watch the following videos:
  PCE Manual Platform Configuration (1 of 4)
  PCE Manual Platform Configuration (2 of 4)
  PCE Manual Platform Configuration (3 of 4)
  PCE Manual Platform Configuration (4 of 4)
  Note: These videos focus on DS-5, but most of the content remains the same for Arm Development Studio. DS-5 is a retired product.

5.4. The ROM Table is incorrect or incomplete

There are several ways a ROM Table can be incorrect or incomplete. Incorrect or incomplete ROM Table(s) can lead to components on the board not being added to the platform configuration. The following is a list of common ROM Table issues:

- If the PRESENT bit is not set for a ROM Table entry, the PCE Console view shows the message Entry present bit not set, no device interrogation will occur. If the PRESENT bit is not set, PCE ignores the ROM Table entry and the corresponding component is not added to the platform configuration.
- If a ROM Table contains the wrong address for a nested ROM Table, the components pointed to by the nested ROM Table are not added to the platform configuration.
- If the ROM Table is not terminated correctly with a 0x0 entry or an entry at ROM Table offset `0xEFC`, the PCE autodetection process might not find all the components on the board.
- If a ROM Table is missing entries for components that are connected to the associated DP or AP, PCE does not add the components to the platform configuration.

If the ROM Table is incorrect or incomplete

1. Compare the components found by the autodetection process to the components listed in your design or board documentation.
2. If the components lists do not match, check the PCE Console view for the message Entry present bit not set, no device interrogation will occur.
   a. If Entry present bit not set, no device interrogation will occur messages are shown, check whether there is a component present at the corresponding ROM Table entry. If there is a component present, correct the PRESENT bit of the ROM Table entry in the hardware design if possible. If correcting the ROM Table entry is not possible, follow the instructions at the end of this section to manually configure a target with PCE and add the missing component.
3. If a subset of the board components is not in the platform configuration, check that all nested ROM Tables addresses are correct. A subset of board components can consist of a core or cores with associated link or trace components or a processor cluster. The ROM Table entry values are shown in the PCE Console view.
   a. If the nested ROM Table address is incorrect, correct the nested ROM Table address in the hardware design if possible. If correcting the nested ROM Table address is not possible, follow the instructions at the end of this section to manually configure a target with PCE and add the missing components.
4. If the platform configuration is missing all the components after a certain point in the autodetection process, in the PCE Console view, check that all the ROM Tables have been read completely and all ROM table reads end with the message End of ROM table.

a. If components are missing because a ROM Table is not read completely or not terminated correctly, correct the ROM Table termination in the hardware design if possible. If correcting the ROM Table is not possible, follow the instructions at the end of this section to manually configure a target with PCE and add the missing components.

5. If any components are still missing from the platform configuration, it might be due to the ROM Table not having an entry for the component. Check your design or board documentation to determine which components are present in each ROM Table. Compare the documented ROM Table component listings to the ROM Table components that are shown in the PCE Console view. If a difference is found, correct the corresponding ROM Table in the hardware design if possible. If correcting the ROM Table is not possible, follow the instructions at the end of this section to manually configure a target with PCE and add the missing component.

Instructions on how to manually configure a target with PCE:

- Consult the **Manual platform configuration section of the Arm® Development Studio User Guide.**
- Consult the **Arm Debugger Manual Configuration Tutorial.**
- Watch the following videos:
  PCE Manual Platform Configuration (1 of 4)
  PCE Manual Platform Configuration (2 of 4)
  PCE Manual Platform Configuration (3 of 4)
  PCE Manual Platform Configuration (4 of 4)
  Note: These videos focus on DS-5, but most of the content remains the same for Arm Development Studio. DS-5 is a retired product.

6 Related information

Here are some resources related to material in this guide:

Common reasons why component and component connections do not appear

Help with connecting to new targets

Help with debugging and tracing targets

How PCE identifies the CoreSight components on the target board

Requesting help with board bring-up