



# Arm Development Studio

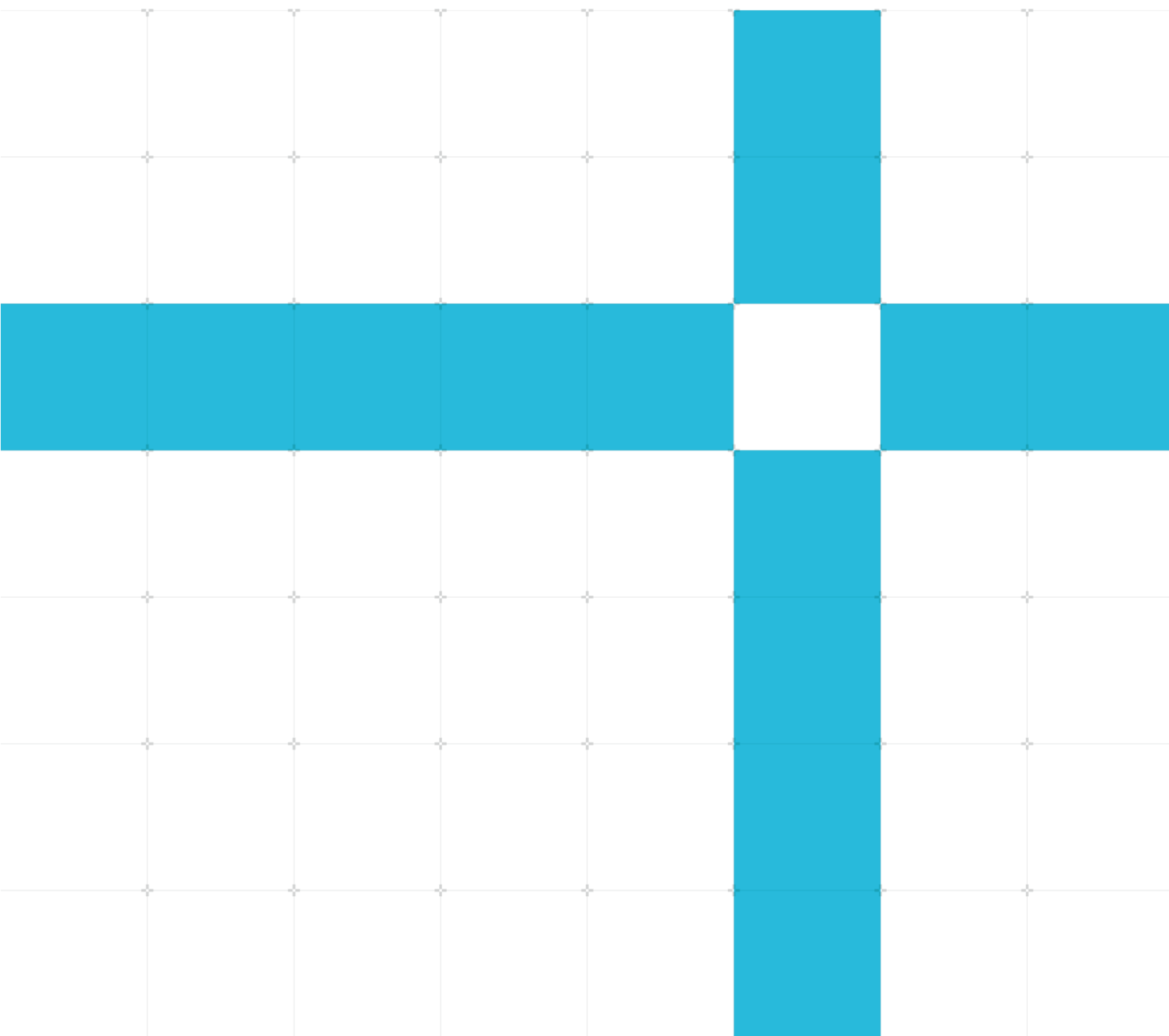
## Debugging with the MCIMX8M-EVK board, DSTREAM-ST, and Arm Development Studio

Non-Confidential

Copyright © 2021 Arm Limited (or its affiliates).  
All rights reserved.

Issue 1.0

102707



## Arm Development Studio Tutorial

### Debugging with the MCIMX8M-EVK board, DSTREAM-ST, and Arm Development Studio

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

#### Release information

#### Document history

Issue	Date	Confidentiality	Change
1.0	November 25, 2021	Non-Confidential	First release

### Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any

conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

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

<b>1 Overview .....</b>	<b>5</b>
1.1 Before you begin .....	5
<b>2 Setting up DSTREAM-ST .....</b>	<b>6</b>
2.1 Install the USB drivers.....	6
2.2 Connect DSTREAM-ST and MCIMX8M-EVK.....	6
2.3 Update the DSTREAM-ST firmware .....	7
<b>3 Create and configure a Hello World project .....</b>	<b>9</b>
3.1 Configure the project to run .....	9
<b>4 Configure a debug connection .....</b>	<b>12</b>
4.1 Disable Android autoboot.....	13
<b>5 Access memory information.....</b>	<b>15</b>
1.1 Use the info memory command .....	15
5.1 Visualize memory maps .....	16
<b>6 Create a platform configuration .....</b>	<b>18</b>
6.1 Create a configuration database .....	18
6.2 Create a new platform configuration.....	18
6.3 Platform Configuration Editor views.....	19
<b>7 Trace .....</b>	<b>23</b>
7.1 Create a project to calculate prime numbers .....	23
7.2 Configure trace .....	24
7.3 Trace view .....	25
7.4 Debug trace.....	26
<b>8 Using CSAT600 with MCIMX8M-EVK board DSTREAM-ST .....</b>	<b>31</b>
<b>9 Related information .....</b>	<b>34</b>

# 1 Overview

In this tutorial, you will learn how to use Arm Development Studio to debug a simple program running on an MCIMX8M-EVK board. By completing a series of basic tasks, you will learn about the different features provided by Arm Development Studio. These tasks include the following:

- Setting up DSTREAM-ST
- Creating and configuring a simple Hello World project
- Configuring a debug connection to the i.MX 8MQuad using DSTREAM-ST
- Using Arm Development Studio to accessing information about memory and the memory map
- Creating a Platform Configuration for the MCIMX8M-EVK board
- Obtaining trace output from the MCIMX8M-EVK board
- Using the CoreSight Access Tool for SoC600 (CSAT600) with the MCIMX8M-EVK board and DSTREAM-ST

## 1.1 Before you begin

This tutorial was written using Arm Development Studio on Windows. If you are using Linux, you will need to adapt some of the instructions.

Before following this tutorial, you should do the following:

- Install and license [Arm Development Studio](#). For more information, see [Arm Development Studio Getting Started Guide](#)
- Obtain a [DSTREAM-ST](#) debug probe
- Obtain an [NXP i.MX 8MQuad Evaluation Kit \(EVK\)](#)

## 2 Setting up DSTREAM-ST

This section describes how to set up the DSTREAM\_ST unit and connect it to the development board. In this section, you will learn how to:

- Install the USB drivers for the DSTREAM-ST unit on Windows
- Connect and power up the DSTREAM-ST unit and the [MCIMX8M-EVK board](#)
- Update the DSTREAM-ST firmware

For further information about the topics in this section, see the [Arm DSTREAM-ST Getting Started Guide](#).

### 2.1 Install the USB drivers

To use the DSTREAM-ST unit with a USB connection, you must [install the USB device drivers](#), which are provided with Arm Development Studio.

These drivers are an optional component of the installation process for Arm Development Studio. If you did not install the USB drivers when you installed Arm Development Studio, follow these steps to install them:

1. Using administrative privileges, run the `driver_install.bat` batch file available in the `<Arm_Development_Studio_install_directory>\sw\driver_files` directory.
2. In the Arm Development Studio Driver Installation Wizard, click **Next** and follow the steps.  
**Note:** During installation, you might receive warnings such as Windows cannot verify the publisher of this driver software. You can safely ignore these warnings and continue with the installation.
3. After the drivers are installed, click **Finish**.

### 2.2 Connect DSTREAM-ST and MCIMX8M-EVK

This section shows how to [connect the DSTREAM-ST to a host PC](#) and the MCIMX8M-EVK board:

1. Connect the DSTREAM-ST probe to your host PC using the 3.0 USB port and the provided USB 3.0 cable.
2. Power up the DSTREAM-ST, using the power adapter for the DSTREAM-ST unit and the power adapter cable appropriate for your region. The LEDs in the DSTREAM-ST follow a [boot sequence](#) when powering up.
3. Connect the DSTREAM-ST to the MCIMX8M-EVK board by plugging the supplied CoreSight 10/20 cable into the [CoreSight 10 connector](#) on the MCIMX8M-EVK board.

4. Power up the MCIMX8M-EVK board and connect it to the host PC using USB. If the target is connected to the DSTREAM-ST unit and powered, the TARGET LED illuminates green, as shown in the following image:



Figure 1: Board connection to the PC

## 2.3 Update the DSTREAM-ST firmware

The DSTREAM-ST firmware is the operating system that runs on the DSTREAM-ST probe. It includes:

- Templates that define debug hardware communication with different targets
- Configuration files to load into any add-on probe units

Firmware updates are supplied with Arm Development Studio. If you get an error message in Arm Development Studio asking you to update the DSTREAM-ST firmware, [update the DSTREAM-ST firmware](#) by following these steps:

1. Open the Debug Hardware Firmware Installer view. From the main menu in Arm Development Studio, select **Window > Show View > Debug Hardware Firmware Installer**, as shown in the following screenshot:

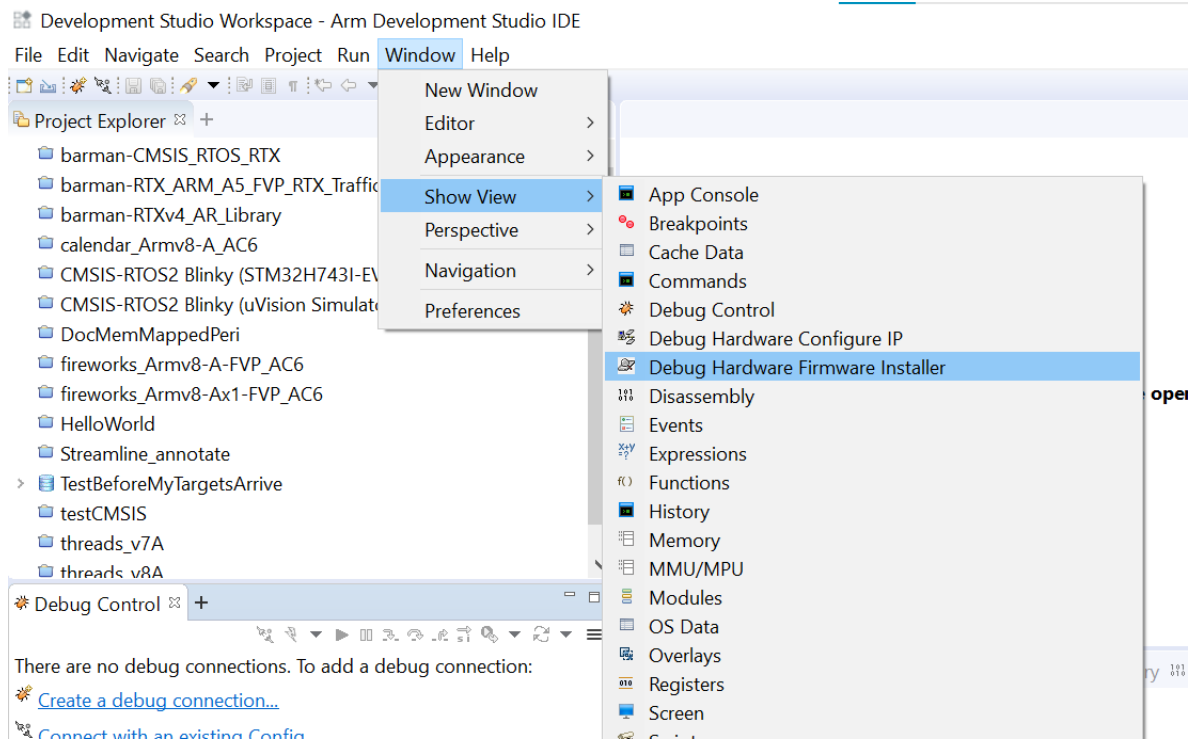


Figure 2: Debug Hardware Firmware Installer option

2. In the **Debug Hardware** option, click **Browse**.
3. Select the DSTREAM-ST unit. Click **Select**.
4. Click **Connect**.
5. Click **Install** to accept the auto-selected firmware update file.

The firmware is now updated on the DSTREAM-ST unit.



## 3 Create and configure a Hello World project

In this section you will learn how to:

- [Create a Hello World ANSI C project](#) using Arm Development Studio
- Configure the project to run on the [i.MX 8MQuad processor](#)

To create a Hello World ANSI C project:

1. Create a new C project by clicking **File > New > Project**.
2. Expand the **C/C++** menu, select **C Project**, then click **Next**.
3. In the **C Project** dialog box:
  - a. In the **Project name** field, enter `iMX8M_Q_Hello_World`.
  - b. Under **Project type**, select **Executable > Hello World ANSI C Project**.
  - c. Under **Toolchains**, select **Arm Compiler 6**.
  - d. Click **Next**.
4. Select **Debug** in the **Configurations** field.
5. Click **Next**. You will see a summary of your project.
6. Click **Finish**.

The project is created and appears in the **Project Explorer** view in Arm Development Studio.

### 3.1 Configure the project to run

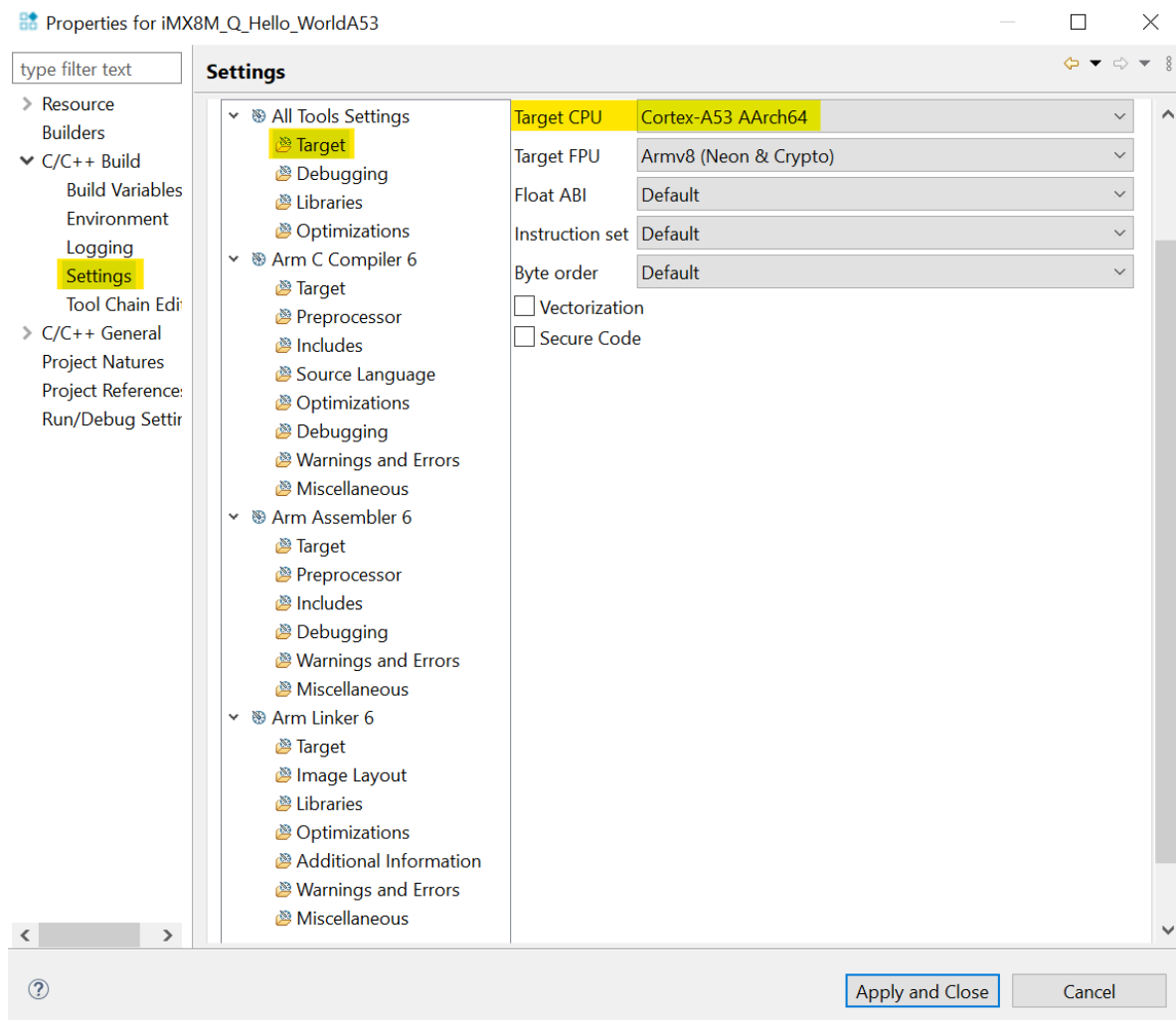
To ensure the application is built and loaded correctly on the target, you must specify the RAM base address to the Arm Linker. You also must set the target CPU. The [i.MX 8MQuad](#) contains four Arm Cortex-A53 (1.5GHz) and one Arm Cortex-M4 (266MHz). If you want to run your program on both Arm Cortex-A53 and on Arm Cortex-M4, you must create two different projects with different settings.

To specify the RAM base address:

1. In the Project Explorer view, right-click the `iMX8M_Q_Hello_World` project and select **Properties**. The Properties for `iMX8M_Q_Hello_World` dialog box opens.
2. Go to **C/C++ Build > Settings**. Select **Arm Linker 6 > Image Layout**.
3. In the **RO base address** field, enter `0x80000000`.
4. Click **Apply and Close**.

To specify the target CPU:

1. In the Project Explorer view, right-click the iMX8M\_Q\_Hello\_World project and select Properties. The Properties for iMX8M\_Q\_Hello\_World dialog box opens.
2. Go to **C/C++ Build > Settings**. Select **All Tools Settings > Target**.
3. Select the processor in the **Target CPU** field. To build and run the program on the **Arm Cortex-M4** processor, select **Generic Armv7E-M**.  
After selecting the target CPU, the other settings in **All Tools Settings > Target** update automatically. Do not change the other fields.
4. To build and run the program on the **Arm Cortex-A53** processor, select **Cortex-A53 AArch64**, as shown in the following screenshot:



**Figure 3: Target CPU menu**

5. Click **Apply and Close**. If you are prompted to rebuild the project, click **Yes**.

After specifying the RAM base address and the target CPU, clean and build the project. You can use the icons in the **Project Explorer** view. Click the broom icon to clean the project and then click the hammer icon to build it.

## 4 Configure a debug connection

In this section you will learn how to [create and configure new hardware connections](#) to debug with the i.MX 8MQuad target, DSTREAM-ST, and Arm Development Studio.

To configure a debug connection:

1. From the Arm Development Studio main menu, select **File > New > Hardware Connection**. Alternatively, you can click **Create a debug connection** in the **Debug Control** view, then select **Hardware Connection** and click **Next**.
2. In the **Hardware Connection** dialog box, specify the details of the connection. In **Debug Connection** enter a debug connection name, for example `imx8mConnection`.
3. Associate the connection with the `iMX8M_Q_Hello_World` project and click **Next**.
4. In **Target Selection** select the **i.MX8M EVK** target. You can type the name of the target to help you find it. You can see the cores in the device and the location of the platform configuration selected. For more information, see [Create a platform configuration](#).
5. Click **Finish** to complete the initial connection configuration.
6. In the **Edit Configuration** dialog box, click the **Connection** tab to specify the target and connection settings.
7. In the **Select target** panel, select a Cortex-M4 or a Cortex-A53 core. Your project must be configured to be built and run in the selected target CPU. For more information, see [Create and configure a Hello World project](#).
8. Select **DSTREAM Family** in the **Target Connection** list.

- To select the connected DSTREAM-ST, click **Browse** in the **Bare Metal Debug Connection** field. After selecting your DSTREAM-ST probe the USB address appears, as shown in the following screenshot:

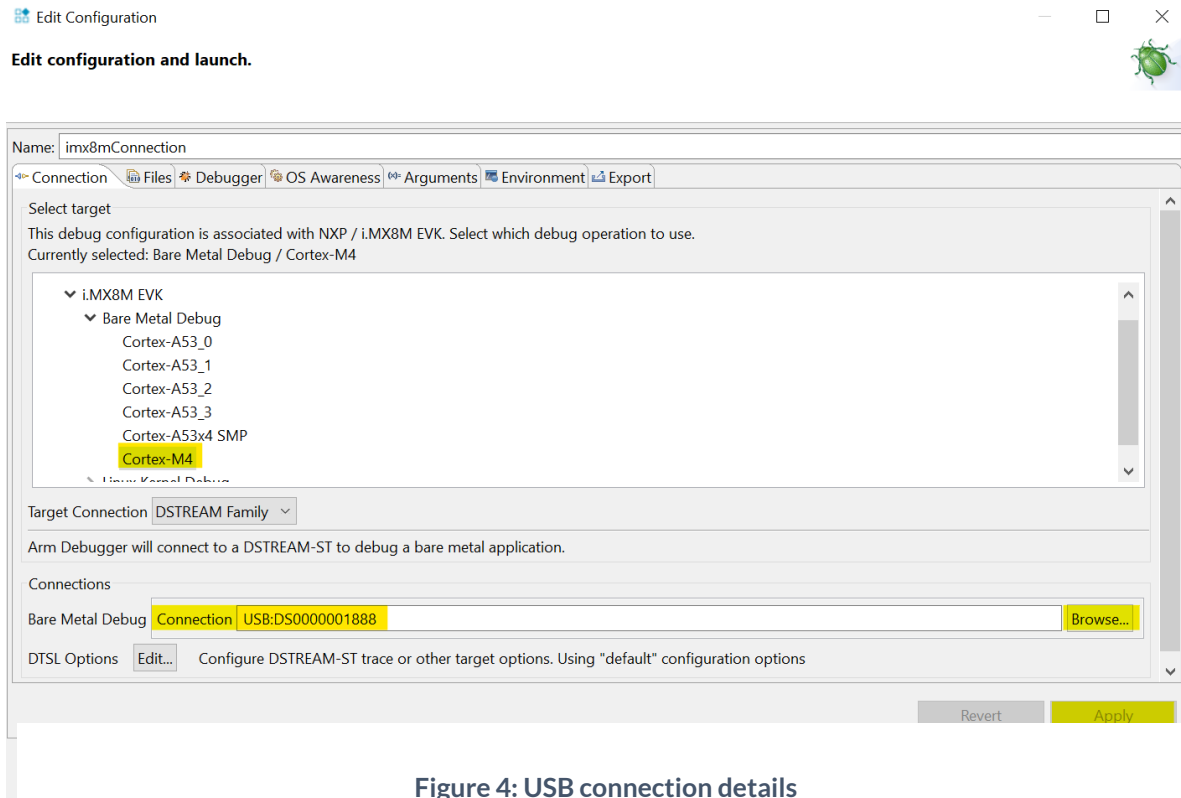


Figure 4: USB connection details

- Click **Apply** to save the changes.
- Click the **Files** tab to specify your application and additional resources to download to the target:
- Click **Workspace** to find and select the application in your workspace. The file `IMX8M_Q_HelloWorld.axf` was generated after building your project in the `debug` folder of the project.
- To debug your application at source level, select **Load symbols**.
- Click **Apply**.
- Click the **Debugger** tab and select **Debug from symbol main**, then click **Apply**.
- To connect to the target and begin debugging, click **Debug**.

## 4.1 Disable Android autoboot

The MCIMX8M-EVK board is shipped with an Android image. For more information, see [Get Started with the MCIMX8M-EVK](#). To connect to the Cortex-A53 cores, you must stop the Android OS from booting on the target.

You can use PuTTY to connect to the serial port and stop Android OS from booting:

1. Start PuTTY.
2. Select the Serial radio button.
3. Set the COM port number (for example COM6, check the Windows Device Manager to know the number).
4. Set a baud rate of 115200.
5. When you connect, hit any key to stop autoboot.

For more information, see [New Hardware Connection](#) and [Help with connecting to new targets](#).

# 5 Access memory information

In this section, you will learn how to:

- Use the [info memory](#) command to display the currently defined memory regions
- Use the [MMU/MPU view](#) to visualize the virtual memory layout and compare it with the memory map in the [i.MX 8MQuad Reference Guide](#)

## 1.1 Use the info memory command

Follow the [Configure a debug connection](#) in this guide before continuing. When you establish the debug connection, you can observe that the program execution is stopped in main. The current instruction in the Disassembly view is shown in the following screenshot:

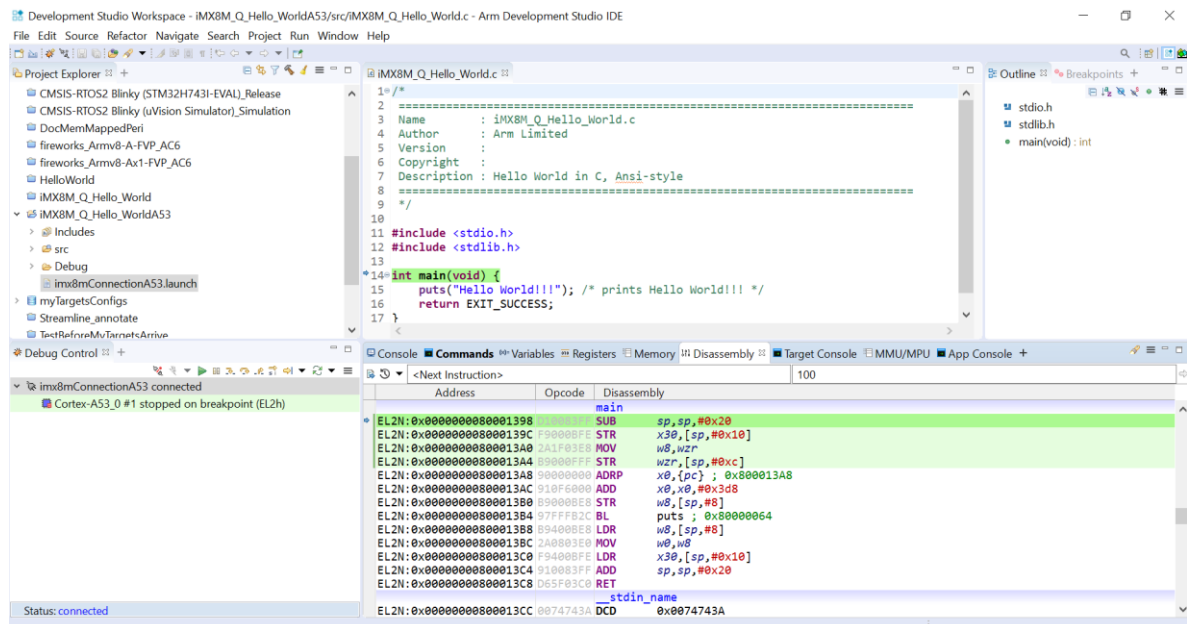


Figure 5: Disassembly view

To display the currently defined memory regions:

1. Click **Window > Show View > Commands** to open the Commands view.
2. Type the command `info memory` or `info mem`. In the Commands view, you can see the different memory regions with their lowest and highest addresses, attributes, and a simple description.

When running the program on a Cortex-A53 core, the Commands view shows the output seen in the following screenshot:

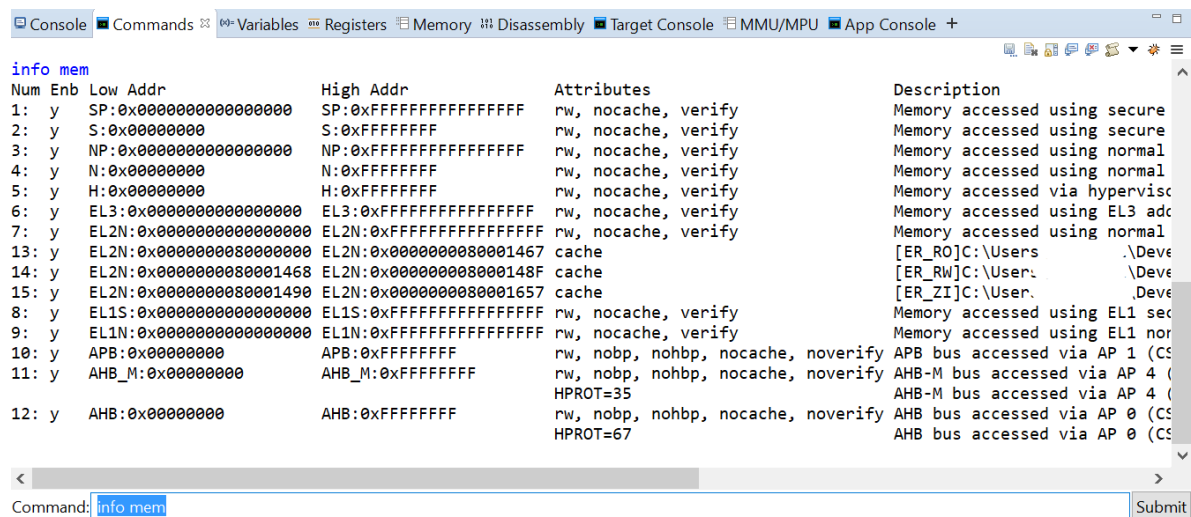


Figure 6: Commands view

You can create a different project to run it on the Cortex-M4 core. The following screenshot shows the output when executing the `info mem` command for a Cortex-M4 project:

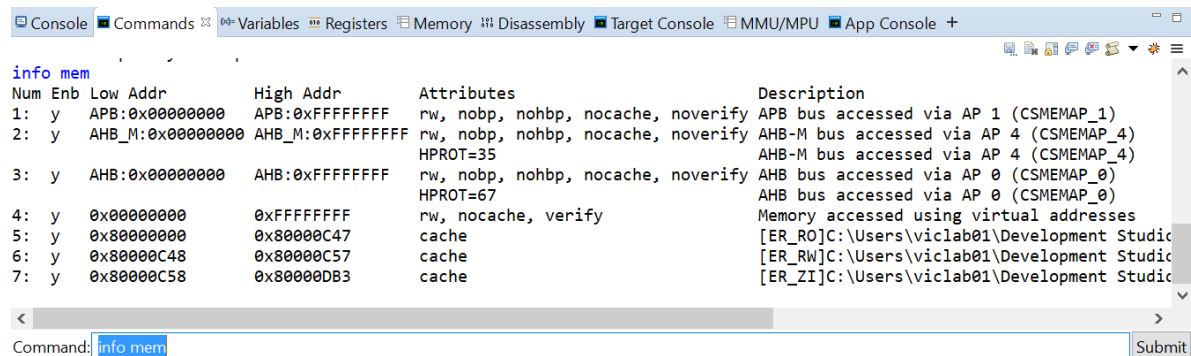


Figure 7: info mem output example

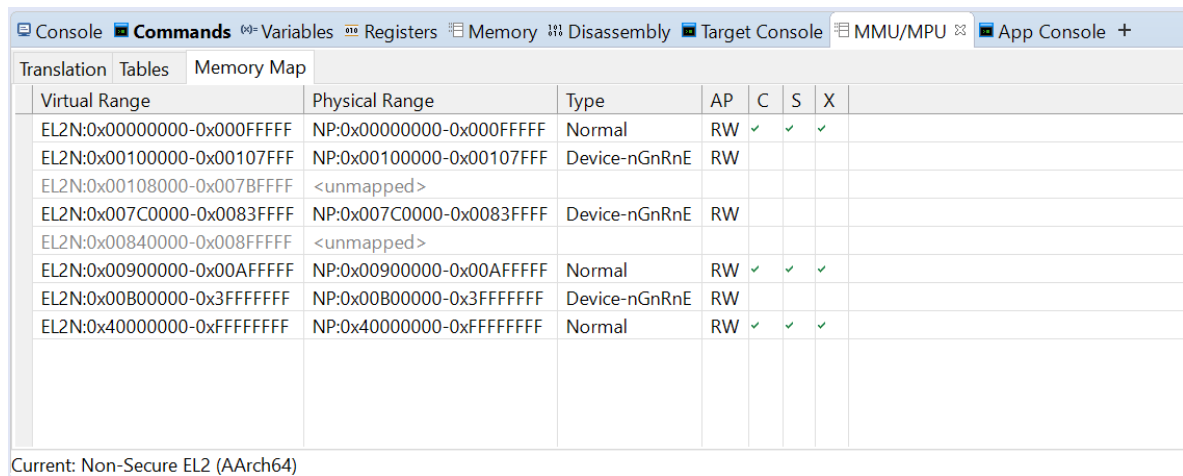
## 5.1 Visualize memory maps

To visualize the memory map when debugging:

1. Click **Window > Show View > MMU/MPU** to open the MMU/MPU view.
2. To visualize the memory map, select the Memory Map tab. The Memory Map tab provides a view of the virtual memory layout. It combines the MMU or MPU table entries that map contiguous regions of memory with a common memory type, for example, cacheability, shareability, and access attributes.



The memory map if you run and debug your program on a Cortex-A53 core is shown in the following screenshot:



Virtual Range	Physical Range	Type	AP	C	S	X
EL2N:0x00000000-0x000FFFFF	NP:0x00000000-0x000FFFFF	Normal	RW	✓	✓	✓
EL2N:0x00100000-0x00107FFF	NP:0x00100000-0x00107FFF	Device-nGnRnE	RW			
EL2N:0x00108000-0x007BFFFF	<unmapped>					
EL2N:0x007C0000-0x0083FFFF	NP:0x007C0000-0x0083FFFF	Device-nGnRnE	RW			
EL2N:0x00840000-0x008FFFFF	<unmapped>					
EL2N:0x00900000-0x00AFFFFF	NP:0x00900000-0x00AFFFFF	Normal	RW	✓	✓	✓
EL2N:0x00B00000-0x3FFFFFFF	NP:0x00B00000-0x3FFFFFFF	Device-nGnRnE	RW			
EL2N:0x40000000-0xFFFFFFFF	NP:0x40000000-0xFFFFFFFF	Normal	RW	✓	✓	✓

Current: Non-Secure EL2 (AArch64)

**Figure 8: Memory map example**

**Note:** Whether an MMU or MPU is implemented depends on the core used. It is not possible to use the MMU/MPU view to visualize the memory map using the Cortex-M4 core.

Compare the Memory Map section in the [i.MX 8MQuad Reference Guide](#) with the information in the **MMU/MPU** view. You can see how they coincide, although the memory map in the Reference Guide is more detailed. For example, the Reference Guide tells us that the region 0x007C0000–0x0083FFFF in the Cortex-A53 memory map corresponds to the Tightly Coupled Memory (TCM).

**Note:** You can also print the memory map using the [mmu memory-map](#) and [mpu memory-map](#) commands in the Command view.

## 6 Create a platform configuration

In this section you will learn how to:

- Create a configuration database to save the platform configurations you create
- Create a new platform configuration for the [MCIMX8M-EVK board](#)
- Discover the different views in the Platform Configuration Editor (PCE)
- Obtain the [CoreSight](#) topology of the MCIMX8M-EVK board from the platform configuration included with the installation of Arm Development Studio

### 6.1 Create a configuration database

Platform configurations must be contained in configuration databases so Arm Development Studio can find the existing platform configurations. Additionally, creating configuration databases helps you to keep the platform configurations you create organized.

To include the platform configuration for the MCIMX8M-EVK board you will create later, you must create a new configuration database:

1. Click **File > New > Other** in the Arm Development Studio main window.
2. Select **Configuration Database** and click **Next**.
3. Give your new configuration database a meaningful name, for example `myTargetsConfigs`.
4. Click **Finish**.

Your new configuration database appears in the **Project Explorer** view.

### 6.2 Create a new platform configuration

To create a new platform configuration for the MCIMX8M-EVK board:

1. Right-click on your Configuration Database in the **Project Explorer** view. For this guide, we use `myTargetsConfigs`. Then click **New > Platform Configuration**.
2. In the **New Platform** window, select the **Import from an existing RDDI configuration file or SDF file (.rcf, .rvc, .sdf), or CoreSight Creator file (.xml):** option. For more information about other New Platform options, see [Create a platform configuration](#).

**Note:** To automatically detect the devices present on the MCIMX8M-EVK board, use the **Automatic/simple platform detection** option. After autodetection, you can add more devices and specify how the devices are interconnected. You must supply the details to connect to the DSTREAM-ST. You can find additional information on how autodetection is performed in [How PCE identifies the CoreSight components on the target board](#). This is the recommended option if you do not have any existing platform configurations files for your target. However, Arm Development Studio installation includes platform configurations for several targets and the

MCIMX8M-EVK board is included. In this case, Arm recommends using the configuration included with Arm Development Studio, using the option **Import from an existing RDDL configuration file or SDF file (.rcf, .rvc, .sdf), or CoreSight Creator file (.xml)**. When using autodetection, Development Studio might not detect all the devices on the platform or might not know how the devices are connected to each other. You can use the [PCE to edit the platform configuration](#) and include the missing devices or connections.

3. Click **Browse** to select the MCIMX8M-EVK configuration included with Arm Development Studio installation. You can find the provided platform configuration file in `C:\Program Files\Arm\Development Studio 2xxx.x\sw\debugger\configdb\Boards\NXP\iMX8M_EVK.sdf`, then click **Next**.

**Note:** You can find the Platform Configuration files included with Arm Development Studio installation in the folder `C:\Program Files\Arm\Development Studio 2xxx.x\sw\debugger\configdb`. The configurations included with Arm Development Studio and the configurations you create appear in the Hardware Connection window when you select a target to establish a debug hardware connection. For more information, see [Configure a debug connection](#).

4. In the **New Platform** window, include the Platform Manufacturer and Platform Name. Click **Finish**.

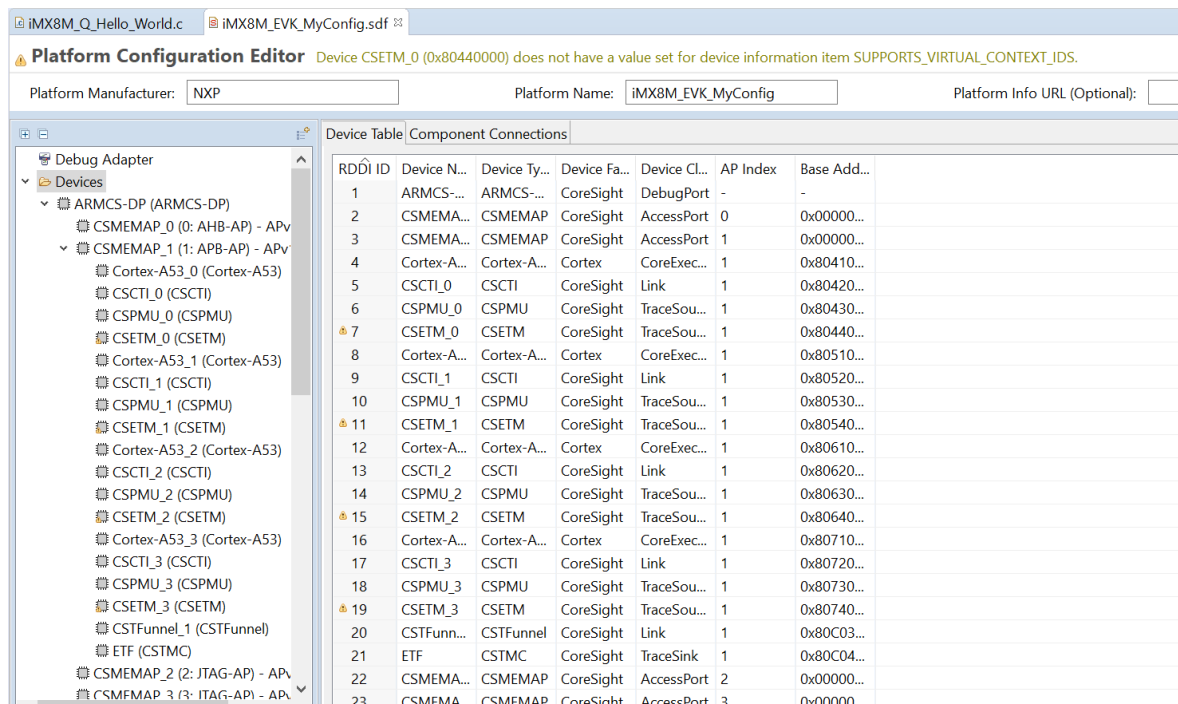
**Note:** You are creating a new platform configuration based on the configuration file for the MCIMX8M-EVK target included in Arm Development Studio. If the platform manufacturer and name are the same as entered in the `configdb` file in the database, the configuration does not appear when you select a target to establish a new hardware debug connection. To add this configuration, use `iMX8M_EVK_MyConfig` in the Platform Name field.

A new platform configuration file `iMX8M_EVK_MyConfig.sdf` is created, which is included in the configuration database `myTargetsConfigs`.

## 6.3 Platform Configuration Editor views

You can use the Platform Configuration Editor (PCE) view to identify missing components and connections and to add them to your platform configuration.

After creating a new platform configuration in Development Studio, you can review it in the PCE. You can see the debug and trace components in your platform configuration in the Device Table. You can add devices, modify the values in their configuration items, and see the details of each of the devices, as shown in the following screenshot:



**Figure 9: PCE window**

In the **Component Connections** tab, you can see, modify, or add connections between the components. The following screenshot shows an example of this tab:

Device Table Component Connections				
Component Connections Help				
Master	Slave	Link Type	Link Details	
CSITM (0xE0000000)	CSTFunnel_0 (0xE0...	ATB	Slave Interface = 1	
CSETM_4 (0xE0041...	CSTFunnel_0 (0xE0...	ATB	Slave Interface = 0	
CSETM_4 (0xE0041...	CSCTI_4 (0xE00440...	CTITrigger	Trigger In = 6	
Cortex-A53_0 (0x80...	CSCTI_0 (0x804200...	CTITrigger	Trigger DBGRESTART = 1	
Cortex-A53_0 (0x80...	CSETM_0 (0x80440...	CoreTrace	N/A	
CSETM_0 (0x80440...	CSCTI_0 (0x804200...	CTITrigger	Trigger In = 6	
Cortex-A53_1 (0x80...	CSCTI_1 (0x805200...	CTITrigger	Trigger DBGRESTART = 1	
Cortex-A53_1 (0x80...	CSETM_1 (0x80540...	CoreTrace	N/A	
CSETM_1 (0x80540...	CSCTI_1 (0x805200...	CTITrigger	Trigger In = 6	
Cortex-A53_2 (0x80...	CSCTI_2 (0x806200...	CTITrigger	Trigger DBGRESTART = 1	
Cortex-A53_2 (0x80...	CSETM_2 (0x80640...	CoreTrace	N/A	
CSETM_2 (0x80640...	CSCTI_2 (0x806200...	CTITrigger	Trigger In = 6	
Cortex-A53_3 (0x80...	CSCTI_3 (0x807200...	CTITrigger	Trigger DBGRESTART = 1	
Cortex-A53_3 (0x80...	CSETM_3 (0x80740...	CoreTrace	N/A	
CSETM_3 (0x80740...	CSCTI_3 (0x807200...	CTITrigger	Trigger In = 6	
Cortex-M4 (0xE000...	CSETM_4 (0xE0041...	CoreTrace	N/A	
Cortex-M4 (0xE000...	CSCTI_4 (0xE00440...	CTITrigger	Trigger DBGRESTART = 7	
CSTFunnel_0 (0xE00...	CSTFunnel_1 (0x80...	ATB	Slave Interface = 2	
CSETM_0 (0x80440...	CSTFunnel_1 (0x80...	ATB	Slave Interface = 0	
CSETM_1 (0x80540...	CSTFunnel_1 (0x80...	ATB	Slave Interface = 0	
Add Link Autodetect Component Connections				

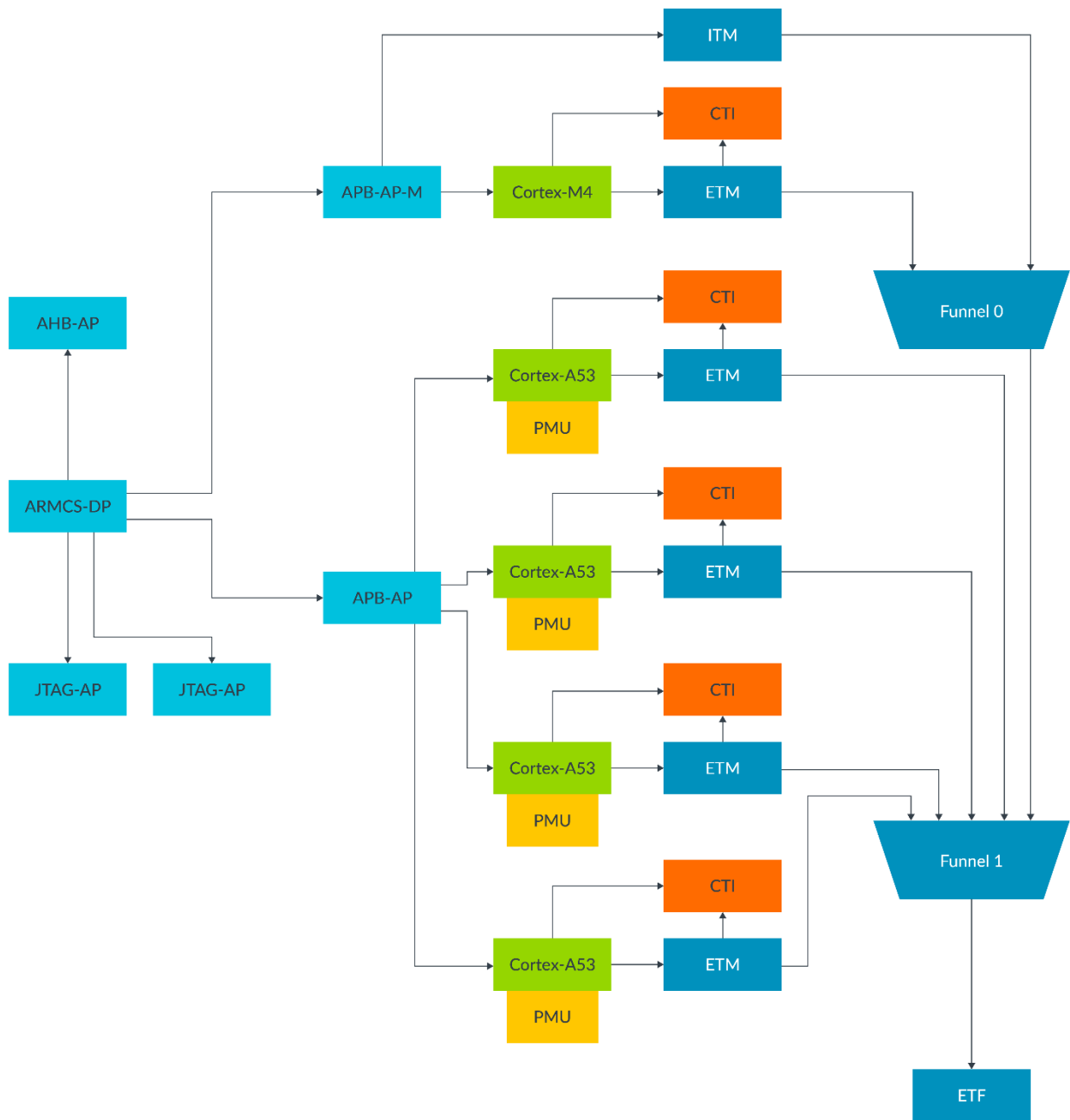
**Figure 10: Component Connections tab**

For more information about how to use the PCE, see [Edit a platform configuration](#).

To use the platform configuration you created and edited, you must build the platform configuration. Right-click the configuration file `IMX8M_EVK_MyConfig.sdf` (which is included in the configuration database `myTargetsConfigs`) in the Project Explorer view and select **Build Platform**.

Use the information in the platform configuration files included with Arm Development Studio and the views in the PCE. This information helps you to create a diagram of the [CoreSight](#) topology for the selected platform. To understand the different CoreSight components, see [Trace components](#) and [Trace infrastructure examples](#).

The following diagram shows the CoreSight topology for the MCIMX8M-EVK board:



**Figure 11: CoreSight topology**

For more information about CoreSight, see [CoreSight Debug and Trace](#).

# 7 Trace

In this section you will learn about:

- [Understanding trace](#) by executing an example code that calculates prime numbers
- [Configuring trace](#) for the MCIMX8M-EVK board using DSTREAM-ST and the Debug and Trace Services Layer (DTSL)
- The information provided in the Trace view
- How to view and select different trace options in the DTSL

## 7.1 Create a project to calculate prime numbers

In this section, you create a new project to run a program that calculates prime numbers in an endless loop.

To create a new project:

1. Follow the steps, properties, and configuration in [Create and configure a Hello World project](#).
2. Create the project `iMX8M_Q_All_Primes_M4` to run on the Cortex-M4 core.
3. Create a second project `iMX8M_Q_All_Primes_A53` to run on the Cortex-A53 core.
4. In both projects, in the `.c` file, replace the code with the following code:

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    unsigned int number;
    int count = 0; //counts the number of prime numbers found

    for(number = 2; number > 0; number ++){

        int isPrime = 1;
        int divisor;

        for(divisor = 2; divisor < number; divisor ++) {
            if((number % divisor) == 0) {
                isPrime = 0;
            }
        }

    }
```

```
        if(isPrime){

            count ++;
            printf("%d is a prime number.\n", number);
            printf("Prime numbers found: %d \n", count);

        }

    }

    return EXIT_SUCCESS;

}
```

5. Name your .c file `iMX8M_Q_All_Primes.c`.

## 7.2 Configure trace

To configure trace using the Debug and Trace Services Layer (DTSL):

1. Create a debug hardware connection by following the instructions in [Configure a debug connection](#).
2. Open the debug configuration file to see the Edit Configuration window. To open this window, double-click on the `.launch` file in the project in the Project Explorer view. Alternatively, with your project selected, select **Run > Debug Configurations** to open the Debug Configurations launcher panel. Then, select the Arm Debugger debug configuration for the MCIMX8M-EVK in the left pane.
3. Select the target in the Connection tab, then click **DTSL Options > Edit**. The DTSL Configuration dialog box where you can configure trace is displayed.
4. Click **+** to create a new trace configuration and name the configuration. For example, to get trace from the Cortex-M4 in the i.MX8MQuad, use the name `myTraceConfig_M4`.
5. Select the Trace Capture tab. In the Trace capture method, select **On Chip Trace Buffer (ETF/ETF)**. The Embedded Trace FIFO (ETF) contains a dedicated SRAM that Arm Development Studio configures as a circular buffer when selecting it as a trace sink.
6. To obtain trace from the Cortex-M4, select the Cortex-M4 tab. Click **Enable Cortex-M4 core trace**. Then click **Enable Cortex-M4 trace** and **Enable ETM Timestamps**. The Embedded Trace Macrocell (ETM) architectures permit instruction trace. Timestamps are useful when calculating how long a code or function takes to execute and for correlating trace data from different sources.
7. In the ITM tab, select **Enable CSITM trace** to enable trace from an Instrumentation Trace Macrocell (ITM). The ITM is a low bandwidth, application-driven trace source mainly used to do the following:
  - o Support printf-style debugging



- Trace OS and application events
  - Output diagnostic system information
8. Click the Events view to see the output generated by the ITM events. Data is captured from your application when it runs. However, no data appears in the Events view until you stop the application.
  9. Click **Apply** after updating the options in any of the tabs. Click **OK**.
  10. Click **Apply** and **Debug** in the **Edit Configuration** window.

## 7.3 Trace view

When you debug with trace enabled, you can find trace information in the Trace view.

To open Trace view:

1. Click **Window > Show View > Trace**.
2. Select the Trace tab. The lower part of the tab shows the trace information obtained by combining trace packets and the source code. You can see the branches that were taken or not and the instructions executed. The upper part of the tab shows a heatmap of the instructions executed. This heatmap shows the percentage of instructions executed in the functions or parts of the code. The following screenshot shows an example of the Trace tab:

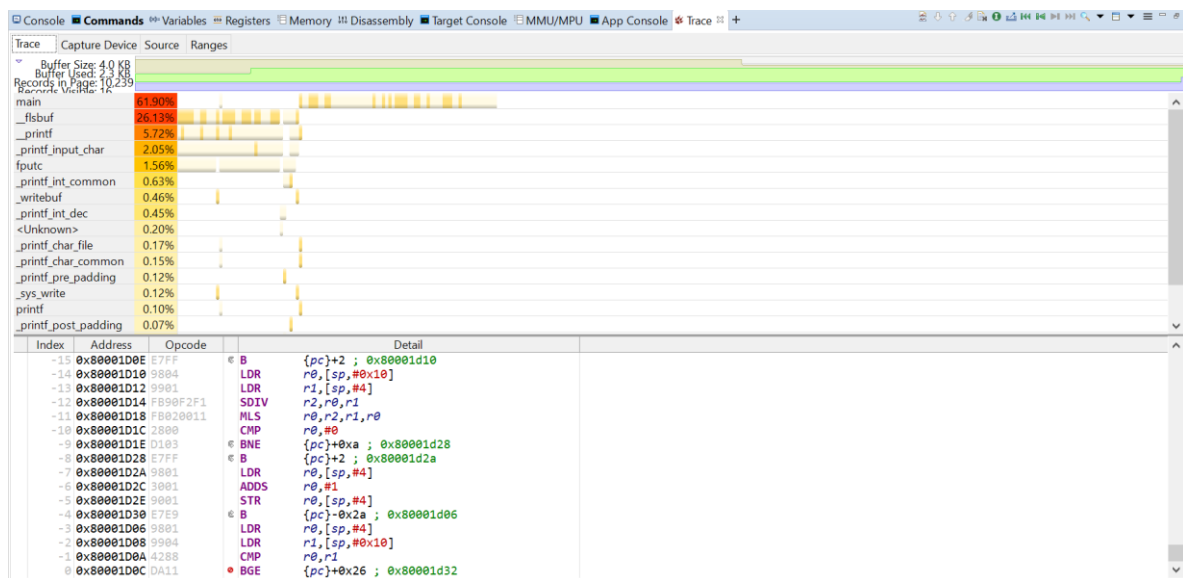


Figure 12: Trace tab

3. Select the Capture Device tab. This tab shows the type of trace capture device used, like an ETF. In this tab, you can select the options for the trace sink, start capture, and clear the trace buffer of the ETF. The Buffer Used and Buffer Size information indicates when the buffer is filling up. The following screenshot shows an example of the Capture Device tab:

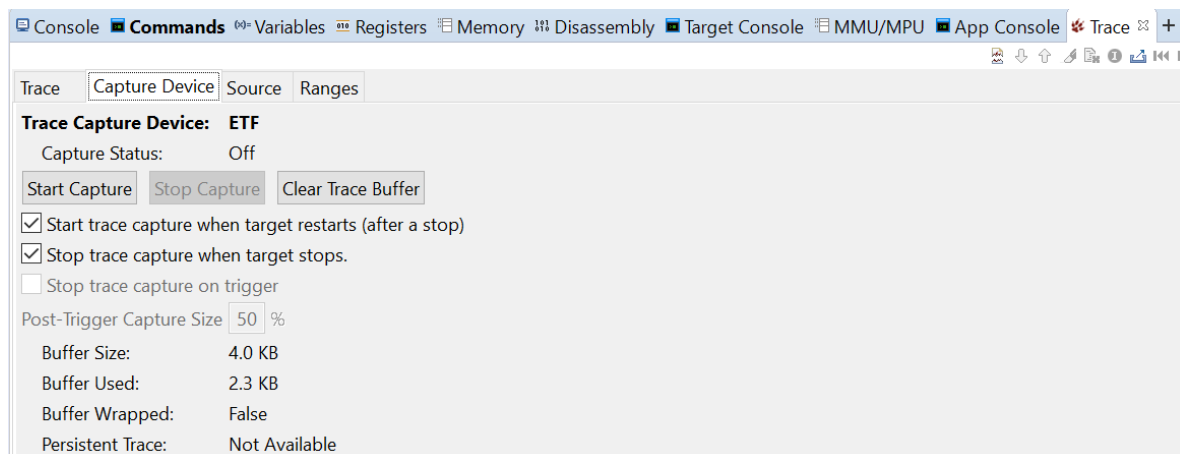


Figure 13: Capture Device tab

4. Select the Source tab. This tab shows the trace source such as an Embedded Trace Macrocell (ETM), the version, and the core. In the following screenshot, ETMv3.5 is used for the Cortex-M4 core:

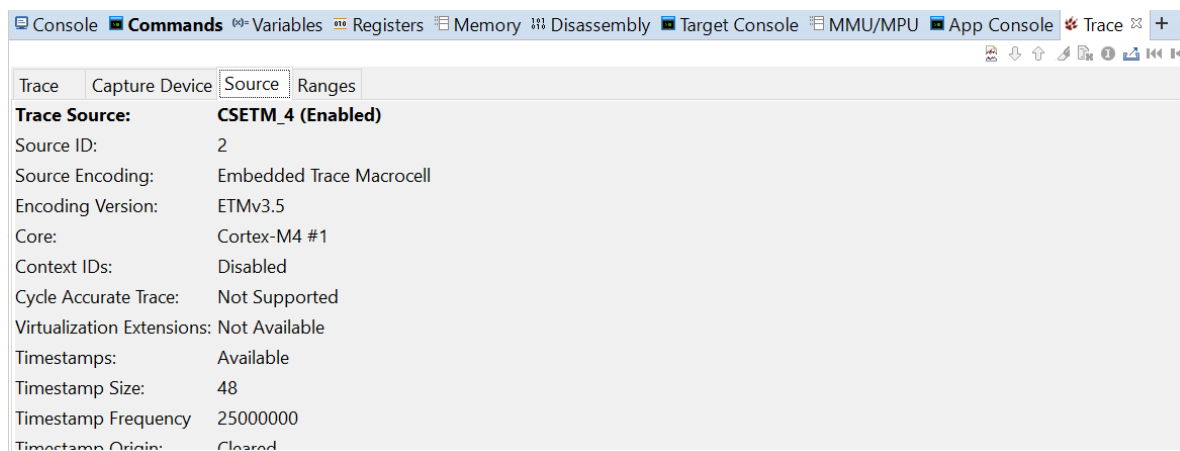


Figure 14: Source tab

The functionality the ETM provides depends on the version and is visible in this tab. For example, Cycle Accurate Trace and Data Trace are not supported by ETMv3.5.

## 7.4 Debug trace

To debug trace on the Cortex-M4 core:

1. Right-click in the line number and click **Toggle breakpoint** to set a breakpoint in the line 26 of `imx8m_q_all_primes.c` (`isPrime = 0;`). The new breakpoint appears in the Breakpoints view.
2. Click **Window > Show View > App Console** to open the App Console view. When you execute the program, the App Console view shows the prime numbers and the number of prime numbers found.
3. Click the green triangle in the Debug Control view until the App Console view shows the output `Prime numbers found: 5`.
4. Click **Window > Show View > Disassembly** to open the Disassembly view. This view shows the assembly instruction where execution halted.
5. Click **Window > Show View > Variables** to open the **Variables** view. Locals shows the following four local variables:
  - o Number contains the current number the program is checking to see whether it is a prime number.
  - o Count contains the total number of prime numbers that have been found since execution began.
  - o IsPrime acts as a boolean, set to zero when a divisor has been found or non-zero.
  - o Divisor contains the current integer the program is checking to see whether it is a divisor for the current number.
6. Click **Window > Show View > Trace** to open the Trace view, then select the Trace tab. The icons at the left of the instructions of the trace view indicate which branches are taken. These branches are generally associated to branch conditions. This view shows a complete history of executed instructions. When you select an instruction in the instruction history, the corresponding line is highlighted in blue in the C code, as shown in the following screenshot:

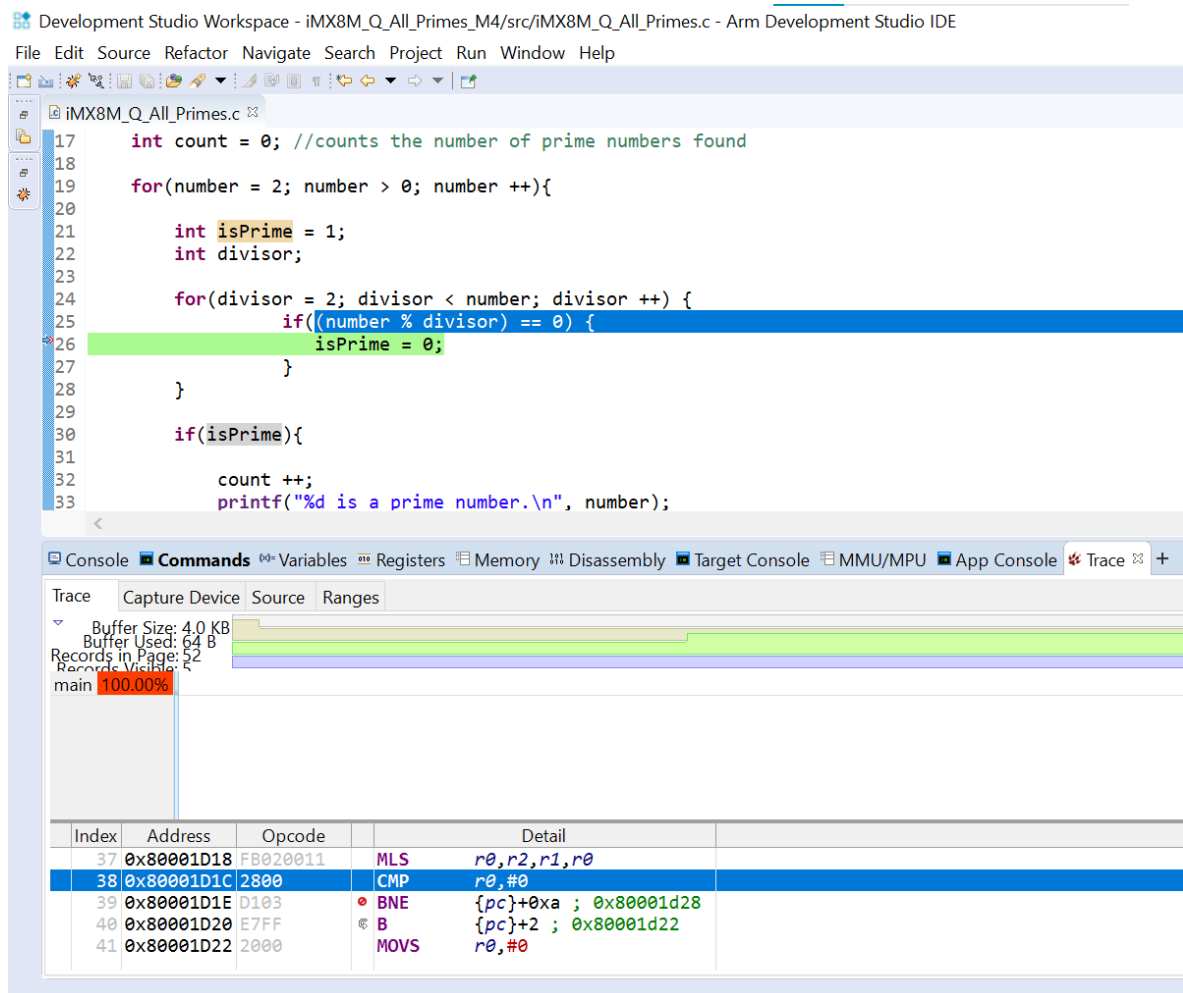
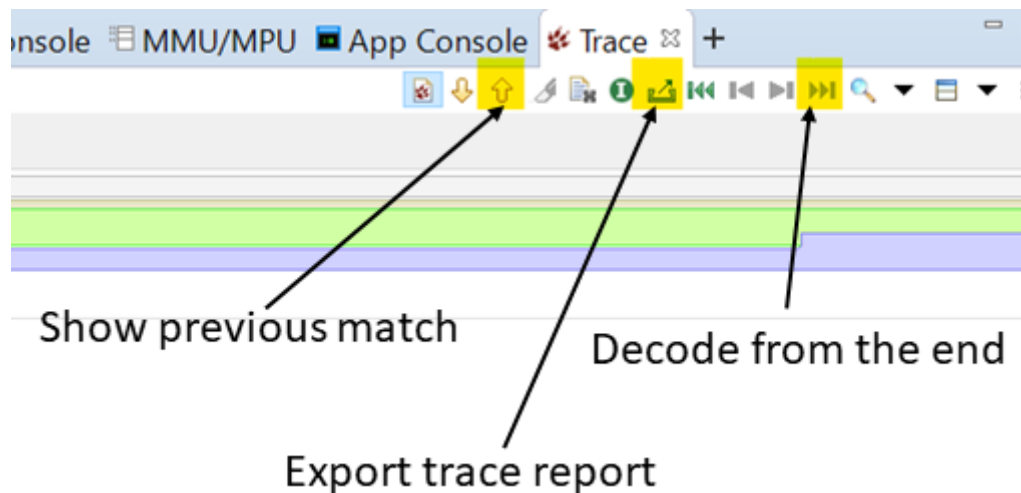


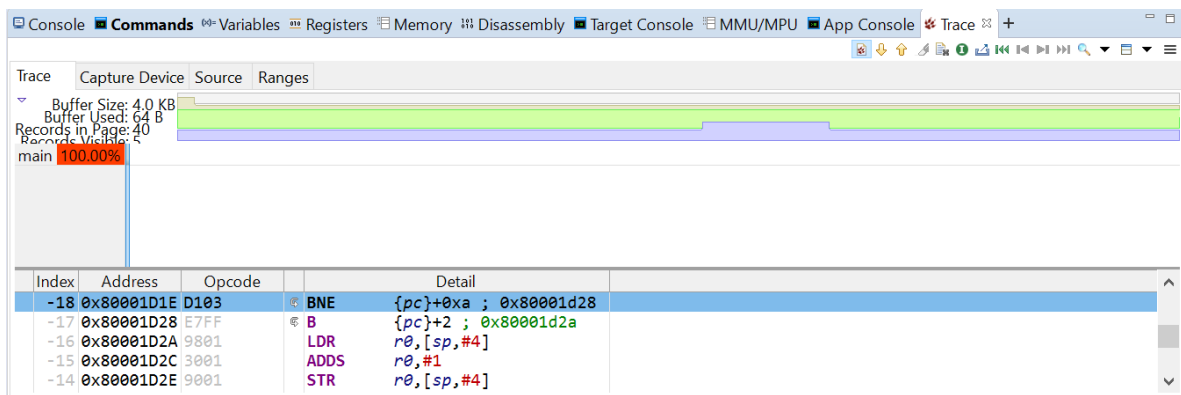
Figure 15: Trace tab

7. Select the Capture Device tab. This tab shows the instruction trace obtained and how many bytes of the ETF buffer are used.
8. Use the icons in the Trace view to perform different options. For example, **Decode from the end** moves the trace view to the end of the trace buffer or you can export a trace report. The Trace controls are shown in the following screenshot:



**Figure 16: Trace controls**

Use **Show previous match** to see when the selected instruction was executed previously in the instruction history. In the following screenshot, the same instruction appears at indexes -2 and -18:



**Figure 17: Show previous match example**

You can also configure trace in the project you created to run on Cortex-A53.

To configure trace in the Cortex-A53 project:

1. Open the DTSL and select the Cortex-A53 tab. Trace for Cortex-A53 cores provides more functionality compared to Cortex-M4. For example, you can enable cycle accurate trace or select trace capture ranges. Trace capture ranges can be useful to, for example, restrict trace to instructions of the kernel of an OS.

- If supported, an ETM can generate cycle-accurate trace. Cycle-accurate trace is useful for determining which code or functions are consuming the most execution time. You can see the cycle-counts added to the instructions in the Trace view, as shown in the following screenshot:

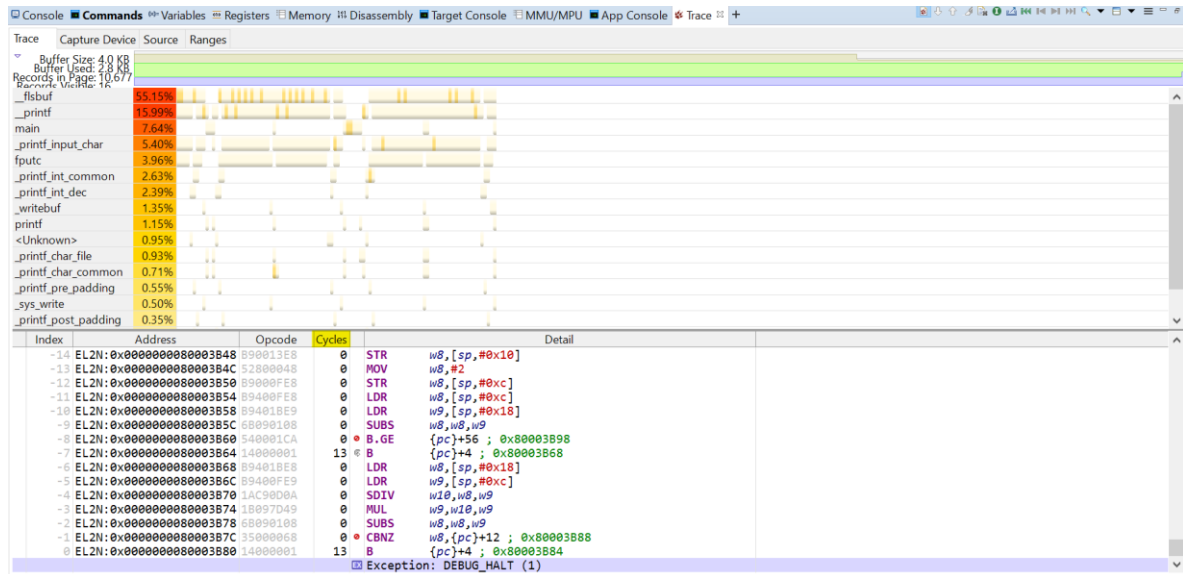


Figure 18: Cycle-count example

**Note:** Including cycle-accurate and timestamping information in the trace data increases the overall trace data size.

- Select the Source tab in the Trace view to see information about your trace source and the functionality it supports. Cortex-A53 cores use ETMv4.0. ETMv4 implementations support cycle accurate trace and timestamps but do not support data trace.

For more information about trace, see the following resources:

- [Configuring trace for bare-metal or Linux kernel targets](#)
- [Help with debugging and tracing targets](#)
- [Learn the architecture: Understanding trace](#)

## 8 Using CSAT600 with MCIMX8M-EVK board DSTREAM-ST

In this section, you learn how to use some of the basic functions of the [CoreSight Access Tool for SoC600 \(CSAT600\)](#). For more information about the CSAT600 commands used in this section and all the commands available, see the [CoreSight Access Tool for SoC600 \(CSAT600\) User Guide](#).

**Note:** The CSAT600 tool is available in Arm Development Studio 2019.0 or later only. For earlier versions of Arm Development Studio, use the [CoreSight Access Tool \(CSAT\) User Guide](#).

In this guide, we use CSAT600 because it is backwards compatible with SoC400 targets. CSAT600 also allows full compatibility and improved functionalities with DSTREAM-ST.

To use CSAT600:

1. Open a new command prompt and go to the folder <Arm Development Studio installation directory>\bin.
2. Run `csat -cs600` to start the CSAT600 tool.  
**Note:** To run the original CSAT tool, run `csat` in an Arm Development Studio bin directory command prompt.
3. Turn on logging to save the output and save the output file as `logTest.txt` to any folder using the following code:

```
log on C:\Users\<User>\csatTest\logTest.txt.
```

4. Connect to the MCIMX8M-EVK using DSTREAM-ST and one of the Platform Configuration .sdf files created in [Create a platform configuration](#). You must specify the USB address for your DSTREAM-ST. In this example, we use the following command:

```
con USB:DS0000001888 C:\Users\<User>\Development Studio  
Workspace\myTargetsConfigs\Boards\NXP\iMX8M_EVK_MyConfig\iMX8M_EVK_MyConfig.sdf
```

5. Read the ROM table with CoreSight devices using the command `list`. Notice that each component has 64kB of memory assigned. For example, in the following output the base addresses 0x80420000 and 0x80430000 correspond to 64kB for the component CSCTI\_0.

```
%> list
```

Device No.	Device Name	Device Type	Base Address	AP No.
0	ARMCS-DP	ARMCS-DP	None	
1	CSMEMAP_0	AHB-AP	N/A	0
2	CSMEMAP_1	APB-AP	N/A	1
3	Cortex-A53_0	Cortex-A53	0x80410000	1
4	CSCTI_0	CSCTI	0x80420000	1
5	CSPMU_0	CSPMU	0x80430000	1

6		CSETM_0		CSETM		0x80440000		1
7		Cortex-A53_1		Cortex-A53		0x80510000		1
8		CSCTI_1		CSCTI		0x80520000		1
9		CSPMU_1		CSPMU		0x80530000		1
10		CSETM_1		CSETM		0x80540000		1
11		Cortex-A53_2		Cortex-A53		0x80610000		1
12		CSCTI_2		CSCTI		0x80620000		1
13		CSPMU_2		CSPMU		0x80630000		1
14		CSETM_2		CSETM		0x80640000		1
15		Cortex-A53_3		Cortex-A53		0x80710000		1
16		CSCTI_3		CSCTI		0x80720000		1
17		CSPMU_3		CSPMU		0x80730000		1
18		CSETM_3		CSETM		0x80740000		1
19		CSTFunnel_1		CSTFunnel		0x80C03000		1
20		ETF		CSTMC		0x80C04000		1
21		CSMEMAP_2		JTAG-AP		N/A		2
22		CSMEMAP_3		JTAG-AP		N/A		3
23		CSMEMAP_4		AHB-AP-M		N/A		4
24		Cortex-M4		Cortex-M4		0xE000ED00		4
25		CSDWT		CSDWT		0xE0001000		4
26		CSFPB		CSFPB		0xE0002000		4
27		CSITM		CSITM		0xE0000000		4
28		CSETM_4		CSETM		0xE0041000		4
29		CSTFunnel_0		CSTFunnel		0xE0043000		4
30		CSCTI_4		CSCTI		0xE0044000		4

6. Connect to a CoreSight device using the device number. To connect to the CSMEMAP\_0 device (AHB-AP device type), use `dvo 1`. After connecting, you can read from and write to the registers of the CoreSight components.
7. Use the command `mr 0x80540000 8` to read the component-specific registers of the CSETM\_1 device. Find its memory mapped address in the list of CoreSight devices from the ROM table shown before. We want to read 8 registers from that address.

```
%> mr 0x80540000 8
Reading from device no. 1: CSMEMAP_0
0x80540000 : 0x04040404
0x80540004 : 0x08081808
0x80540008 : 0x08080808
0x8054000c : 0x08080808
0x80540010 : 0x08100810
0x80540014 : 0x0404050c
```



```
0x80540018 : 0x10080504  
0x8054001c : 0x08080808
```

8. Write a value in a component specific register. For example, to write the value 0x10080504 in the register in the address 0x80540018, use the command `mw 0x80540018 0x10080504`.
9. Read the register in the address 0x80540018 to check the value is correctly set. Use the command `mr 0x80540018 1`. The number 1 means you only one to read one position in memory.
10. Disconnect from the CSMEMAP\_0 device using the command `dvc`. Disable logging using `log off`.
11. Disconnect from the DSTREAM-ST using `dcn` and exit CSAT600 using the command `exit`.

For more information about CSAT600, see the following resources:

- [CoreSight Access Tool for SoC600 \(CSAT600\) User Guide](#)
- [CoreSight Access Tool \(CSAT\) User Guide](#)
- [How to use CSAT CoreSight Components](#)
- [Understanding the CoreSight DAP](#)

## 9 Related information

The following material is related to information in this guide:

- [Arm Development Studio Getting Started Guide](#)
- [Arm DSTREAM-ST Getting Started Guide](#)
- [Configuring trace for bare-metal or Linux kernel targets](#)
- [CoreSight Access Tool \(CSAT\) User Guide](#)
- [CoreSight Access Tool for SoC600 \(CSAT600\) User Guide](#)
- [Help with connecting to new targets](#)
- [Help with debugging and tracing targets](#)
- [How to use CSAT CoreSight Components](#)
- [Learn the architecture: Understanding trace](#)
- [New Hardware Connection](#)
- [Understanding the CoreSight DAP](#)