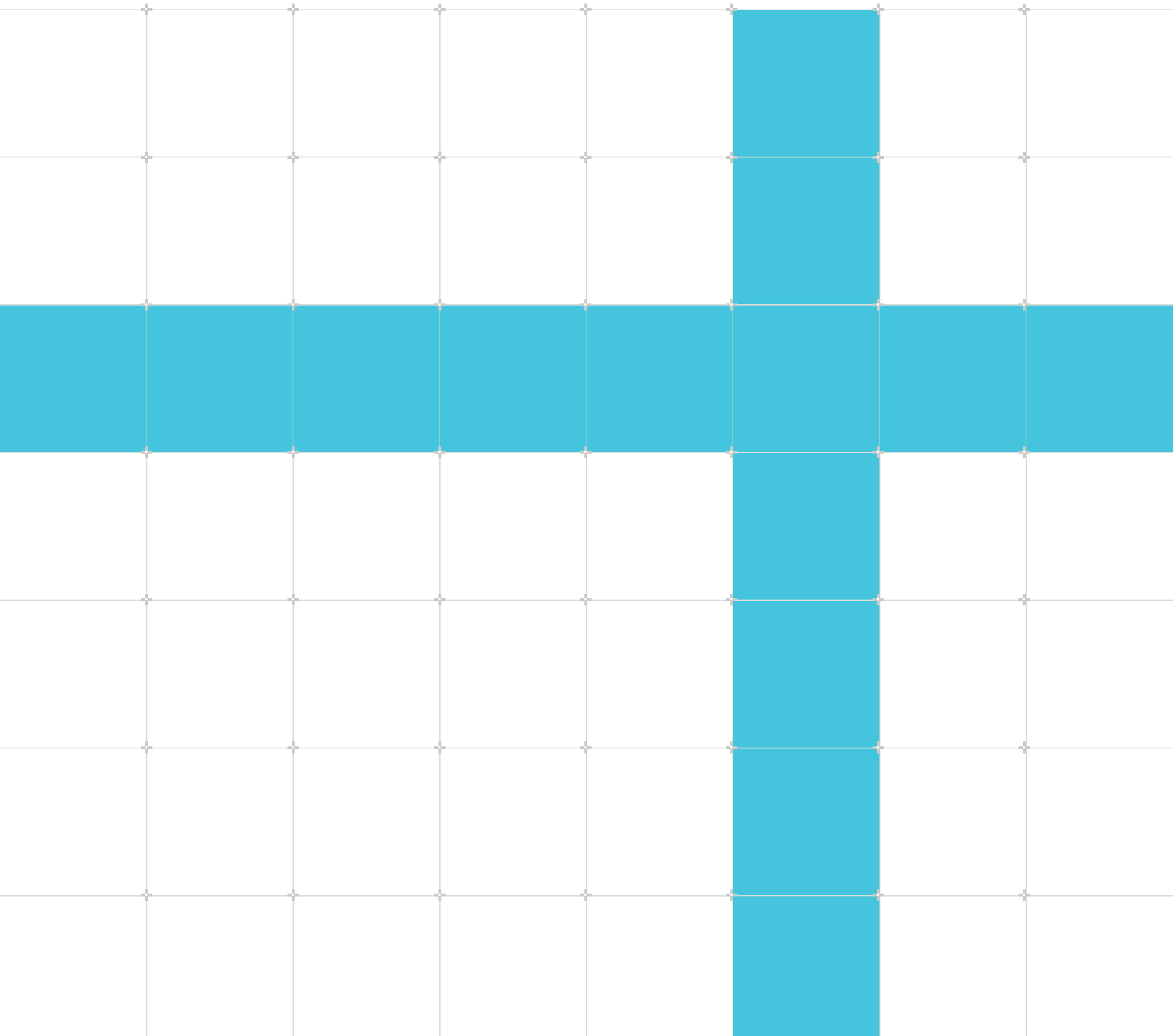
Arm® Development Studio

Version 2023.1

Hello World tutorial

Issue 01
109345_2023.1_01_en



Arm® Development Studio

Hello World tutorial

Release information

Document history

Issue	Date	Confidentiality	Change
2023.1-01	18 February 2025	Non-Confidential	New document

Proprietary Notice

makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

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.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted 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.

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 this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

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

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

Inclusive language commitment

Inclusive language commitment

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

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

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

Contents

1. Introduction

The Hello World tutorial is for new users of Arm® Development Studio, taking you through each step in getting your first project up and running. This tutorial uses Arm Development Studio 2023.1.

1.1 Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

We recommend the following. If you do not follow these recommendations your system might not work.

Your system requires the following. If you do not follow these requirements your system will not work.

You are at risk of causing permanent damage to your system or your equipment, or harming yourself.

This information is important and needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

A reminder of something important that relates to the information you are reading.

1.2 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm Debugger Command Reference	101471	Non-Confidential
Arm Development Studio Getting Started Guide	101469	Non-Confidential
Arm Development Studio User Guide	101470	Non-Confidential

1.3 Other information

See the Arm website for other relevant information.

- Arm® Developer.
- Arm® Documentation.
- Technical Support.
- Arm® Glossary.

2. Open Arm Development Studio for the first time

The first time you open Arm® Development Studio, you are prompted to add your license details. When you have completed the tasks in this section, you are ready to use Arm Debugger.

Before you begin

- Check that your operating system meets the Arm Development Studio system requirements. For details see Hardware and host platform requirements.
- Download and install Arm Development Studio on your operating system with one of these options:
  - Linux: Install Arm Development Studio on Linux
  - Windows using the command-line: Install Arm Development Studio on Windows using the command line
  - Windows using the installation wizard: Install Arm Development Studio on Windows using the installation wizard
- If you or your company has purchased Arm Development Studio, you need one of the following:
  - Arm user-based licensing: The license server address or an activation code.
  - FlexNet license management: The license file or the license server address and port number.

Procedure

1. Open Arm Development Studio:
   - On Windows, select **Windows menu > Arm Development Studio <version>**.
   - On Linux:
     - GUI: Use the menu system of your Linux variant to locate Arm Development Studio.
     - Command line: Run `<installation_directory>/bin/armds_ide`
2. The first time you open Arm Development Studio, the **Product Setup** dialog box opens, which prompts you to add your product license. You can select one of the following:
   - **Manage Arm User-Based Licenses** - select this option if you have purchased Arm Development Studio and it is licensed using Arm user-based licensing. After selecting this option, click **Finish** to open the **Arm License Management Utility** dialog box.


Note

Arm user-based licensing is only available to customers with a user-based licensing license. Documentation for user-based licensing is available at https://lm.arm.com. For assistance with user-based licensing issues, visit https://developer.arm.com/support and open a support case.

- **Add product license** - select this option if you have purchased Arm Development Studio and it is licensed by FlexNet licence management. After selecting this option:
  a. Click **Next**.
  b. Enter the location of your license file, or the address and port number of your license server, and click **Next**.
  c. The Arm Development Studio editions that you are entitled to use are listed. Select the edition that you require, and click **Next**.
  d. Check the details on the summary page. If they are correct, click **Finish**.
- **Obtain evaluation license** - select this option if you would like to evaluate the product. After selecting this option:
  a. Click **Next**.
  b. Log into your Developer account using your Arm Developer account email address and password. If you do not have an account, click **Create an account**.
  c. Select a network interface to which your license will be locked.
  d. Click **Finish**.

Results

Arm Development Studio opens. The main features of the user interface are described in the IDE Overview.


Note

The workspace is automatically set by default, to either:

- Windows: `<userhome>\Development Studio Workspace`
- Linux: `<userhome>/developmentstudio-workspace`

You can change the default location by selecting **File > Switch Workspace**.

---

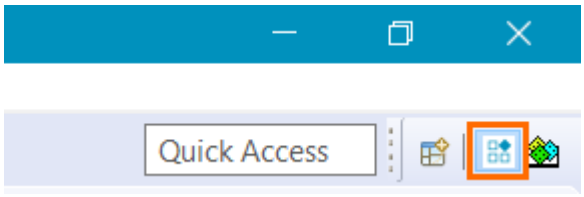
3. Create a project in C or C++

After installing and licensing Arm® Development Studio, we are going to create a simple Hello World C project and show you how to specify the base RAM address for a target. For the remainder of this tutorial, we are going to use the Arm Compiler for Embedded 6 toolchain and our target is a Cortex®-A53 Fixed Virtual Platform, provided with Arm Development Studio.

Before you begin

- Complete Open Arm Development Studio for the first time.
- Ensure you are viewing the **Development Studio** perspective of Arm Development Studio. This perspective is the default perspective when you first open Arm Development Studio. To return to this perspective, click **Development Studio** in the top right corner.

Figure 3-1: Screenshot highlighting the button for the Development Studio Perspective



Procedure

1. To create a new C project, select **File > New > Project**.
2. Expand the **C/C++** menu, and select **C project**, then click **Next**.

Note

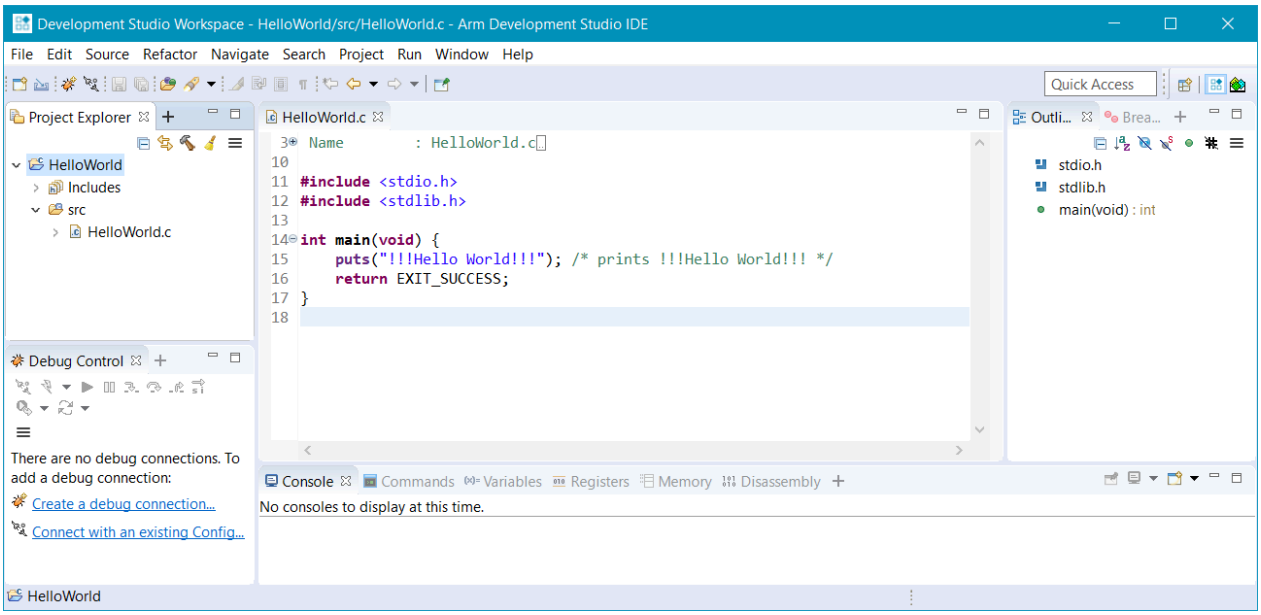
This tutorial also works with a C++ project.

---

3. In the **C Project** dialog box:
   a) Enter `HelloWorld` in the **Project name** field.
   b) Under **Project type**, select **Executable > Hello World ANSI C Project**.
   c) Under **Toolchains**, select **Arm Compiler for Embedded 6**.
   d) Click **Finish**.

Results

Figure 3-2: The IDE after creating a new project



Next steps

To add existing source files to your project, drag and drop the file into the project folder. Alternatively, select **File > Import > General > File System**.

4. Configure your project

Before you build the `HelloWorld` project, you must specify some configuration settings.

Before you begin

Complete Create a project in C or C++.

About this task

You must specify:

- The target processor or architecture you want to compile for.
- That the compiler must add debug symbols into the image file, so that the debugger can debug it at source-level.
- The address in RAM in your FVP target where you want the linker to base your image.

This ensures that the application is built and loaded correctly on to your target, and that you can debug the image at source-level.

Procedure

1. In the **Project Explorer** view, right-click the `HelloWorld` project and select **Properties**. The **Properties for HelloWorld** dialog box opens.
2. Add debug symbols into the image file:
   a) Expand **C/C++ Build**, and select **Settings**.
   b) Ensure the **Configuration** is set to **Debug [Active]**.
3. Configure the target:
   a) Select **C/C++ Build > Settings**.
   b) In **Tool Settings** tab, select **All Tools Settings > Target**.
   c) From the **Target CPU** dropdown, select **Cortex-A53 AArch64**.
   d) From the **Target FPU** dropdown, select **Armv8 (Neon)**.
4. Configure the image layout:
   a) In the **Tool Settings** tab, select **Arm Linker 6 > Image Layout**.
   b) Enter `0x80000000` in the **RO base address** field.
5. Click **Apply and Close**.
6. If you are prompted to rebuild the index, click **Yes**.

5. Build your project

You can now build your `HelloWorld` project.

Before you begin

Complete Configure your project.

Procedure

In the **Project Explorer** view, right-click the `HelloWorld` project and select **Build Project**.

Results

When the project has built, in the **Project Explorer** view, under **Debug**, locate the `HelloWorld.axf` file.

The `.axf` file contains the object code and debug symbols that enable Arm® Debugger to perform source-level debugging.

Note

Debug symbols are added at build time. You can either specify this manually, using the `-g` option when compiling with Arm Compiler for Embedded 6, or you can set this to be default behavior. See Configure your project for details.

6. Configure your debug session

In Arm® Development Studio, you configure a debug session with the **New Debug Connection** wizard. This wizard enables you to connect to your target.

About this task

Depending on your requirements, you can:

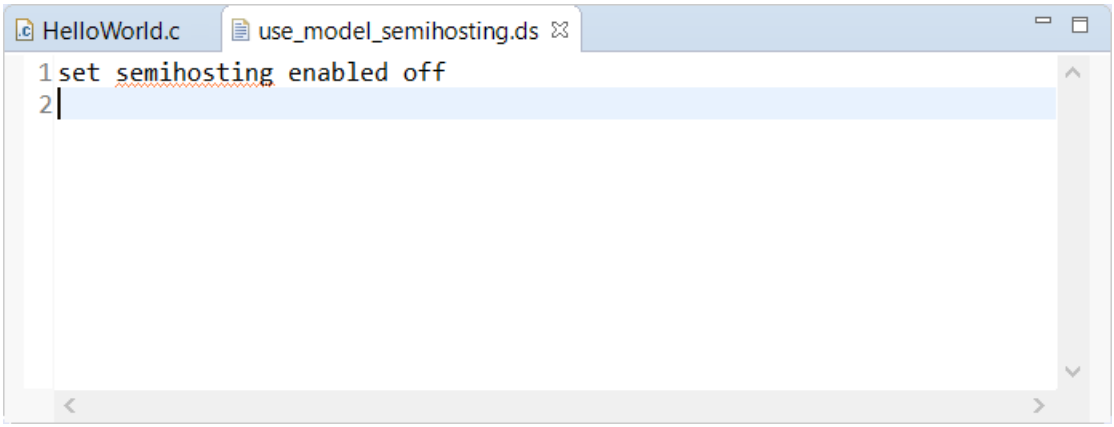
- Configuring a connection to an external FVP for bare-metal application debug
- Configuring a connection from the command-line to a built-in FVP
- Configuring a connection to a Linux application using gdbserver
- Configuring a connection to a Linux kernel

The following example takes you through configuring a bare-metal **Model Connection** to a Cortex®-A53 Fixed Virtual Platform (FVP), using the project you created in Build your project.

Procedure

1. Create a `.ds` script so that the FVP handles semihosting, instead of Arm Debugger:
   a) From the main menu, select **File > New > Other**.
   b) In the **Select a wizard** dialog box, select **Arm Debugger > Arm Debugger Script** and click **Next**.
   c) Click **Workspace** and select the **HelloWorld** project as the location for this script. Click **OK**.
   d) In the **File Name** field, name this script `use_model_semihosting` and click **Finish**. The empty script opens in the **Editor** window.
   e) Add the following code to the script and press **Ctrl + S** to save:
   ```
   set semihosting enabled off
   ```

   Figure 6-1: Editor window with semihosting script

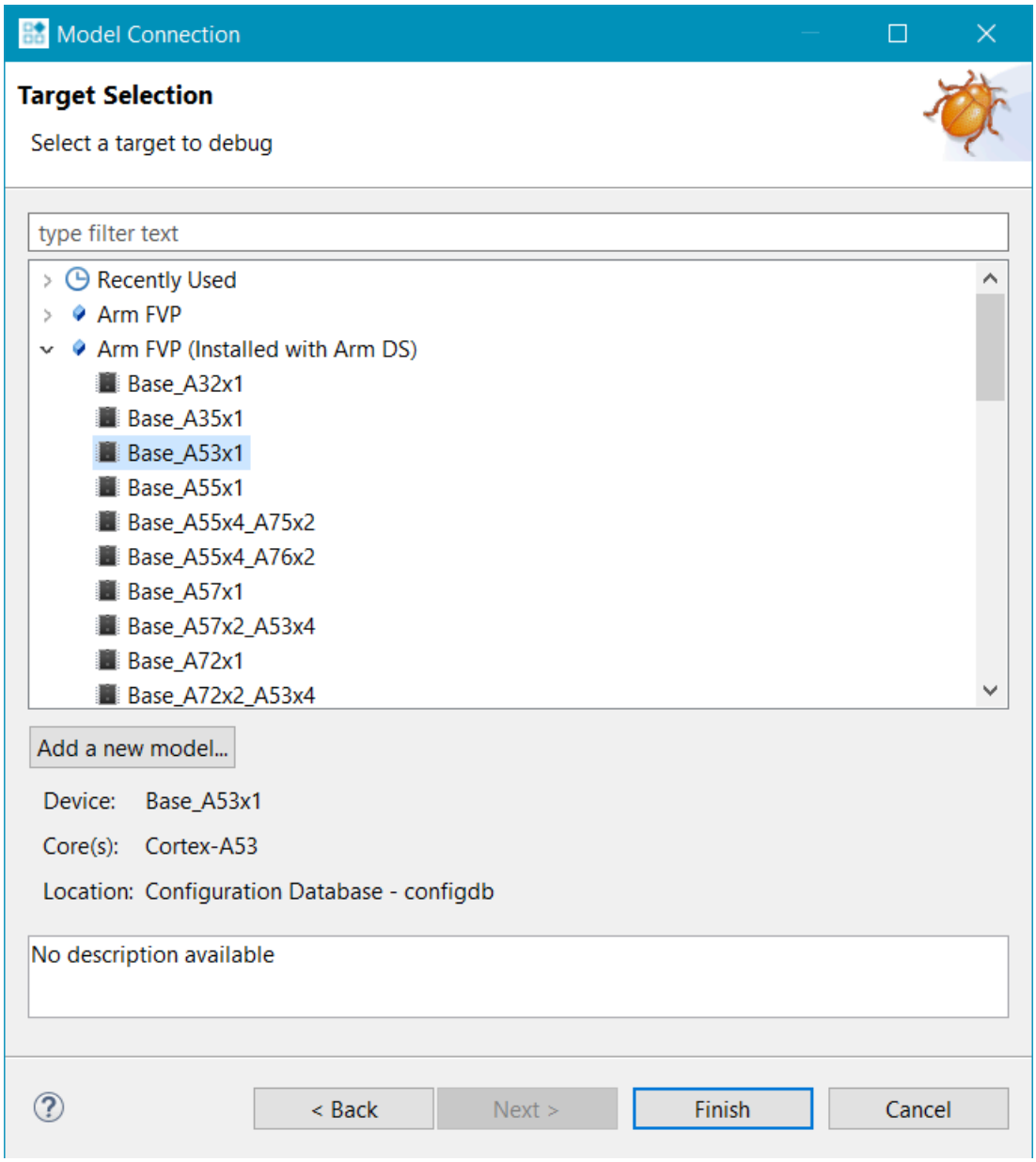


2. From the main menu, select **File > New > Model Connection**.
3. In the **Model Connection** dialog box, specify the details of the connection:
   a) Enter a name for the debug connection, for example **HelloWorld_FVP**.
   b) Select **Associate debug connection with an existing project**, and select the project that you created and built in the previous section Build your project.

c) Click **Next**.

4. In the **Target Selection** dialog box, specify the details of the target:
   a) Select **Arm FVP (Installed with Arm DS) > Base_A53x1**.

Figure 6-2: Select Base_A53x1 model



   b) Click **Finish**.

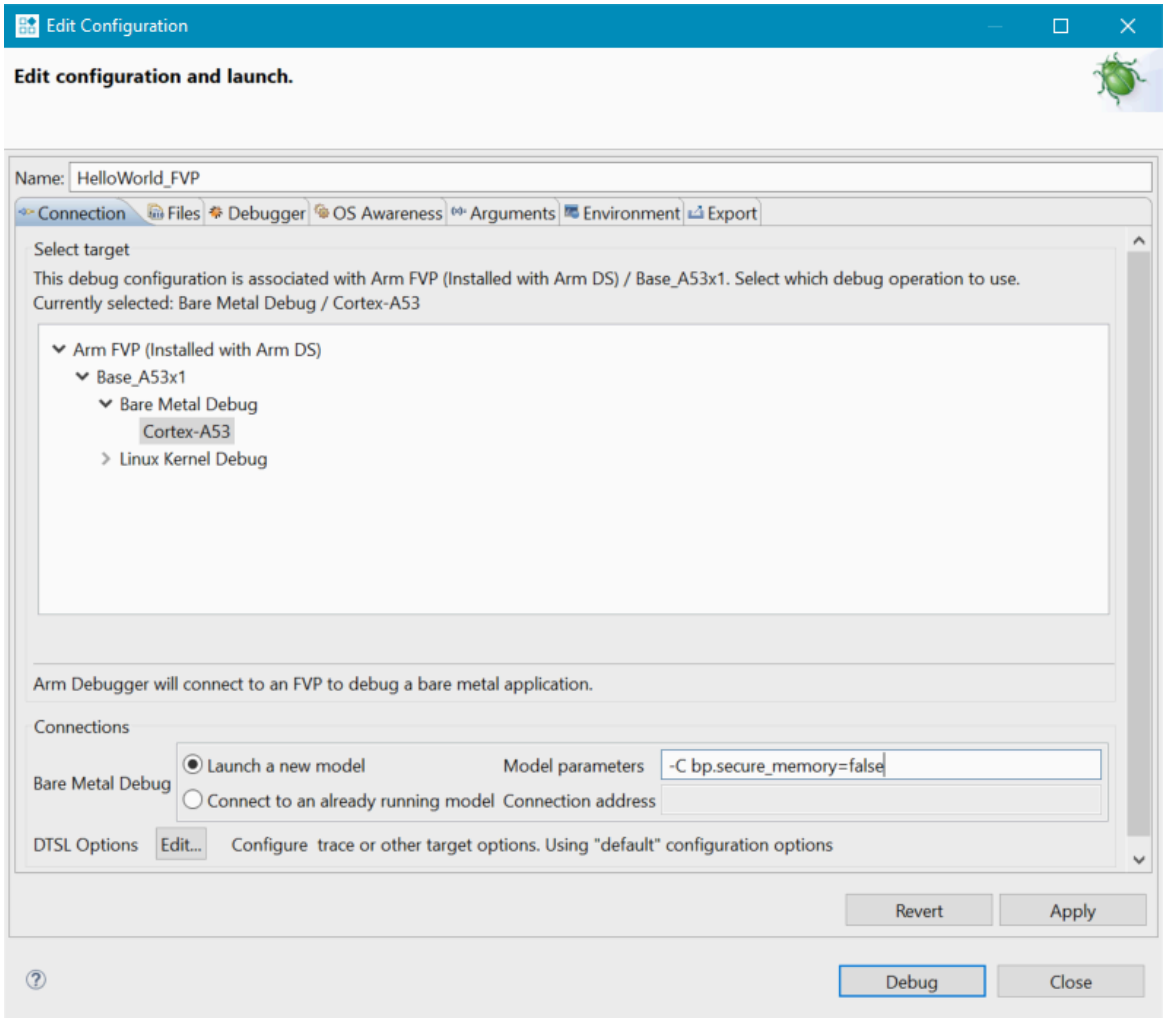
5. In the **Edit Configuration** dialog box, ensure the right target is selected, the appropriate application files are specified, and the debugger knows where to start debugging from:
   a) Under the **Connection** tab, ensure that **Arm FVP (Installed with Arm DS) > Base_A53x1 > Bare Metal Debug > Cortex-A53** is selected.
   b) Under **Bare Metal Debug**, in the **Model parameters** field, add the following parameter:
   `-C bp.secure_memory=false`

Note

For Cortex-M models, the parameter to add is `-c fvp_mps2.DISABLE_GATING=1`.

This parameter disables the TZC-400 TrustZone memory controller included in the Base_A53x1 FVP. By default, the memory controller refuses all accesses to DRAM memory.

Figure 6-3: Edit configuration Connection tab



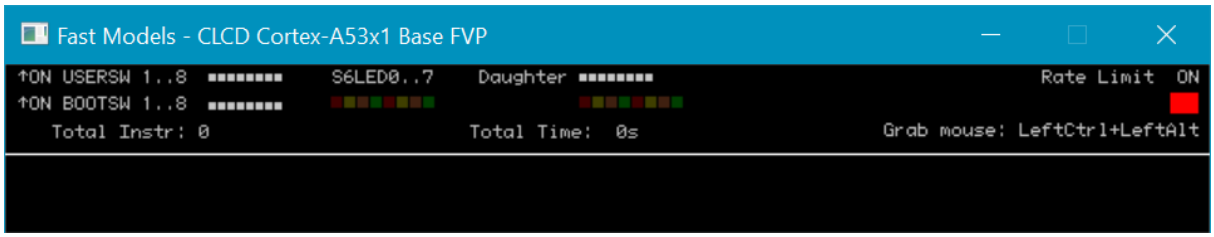
c) In the **Files** tab, select **Target Configuration > Application on host to download > Workspace**.
d) Click and expand the **HelloWorld** project and from the **Debug** folder, select `HelloWorld.axf` and click **OK**.
e) In the **Debugger** tab, select **Debug from symbol**.
f) Enable **Run target initialization debugger script (.ds/.py)** and click **Workspace**.
g) Select the `use_model_semihosting.ds` script and click **OK**.

6. Click **Debug** to load the application on the target, and load the debug information into the debugger.

Results

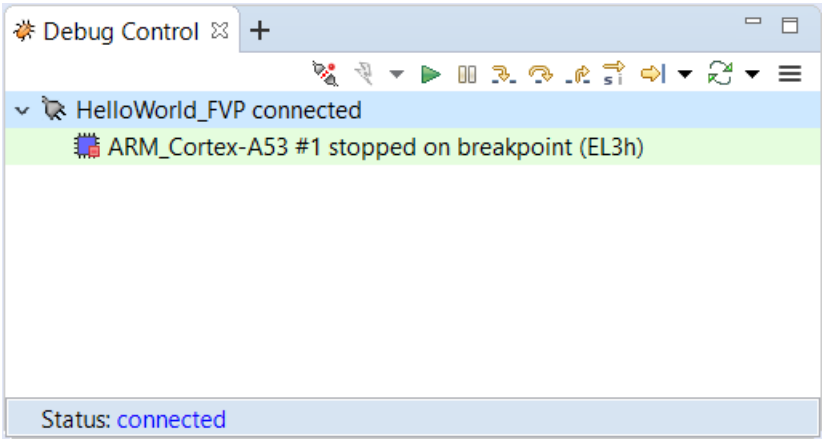
- By default for FVPs, the **CLCD window** launches. You can disable this default action with the `-C bp.vis.disable_visualisation=1` parameter. See Using the CLCD window for more information.

Figure 6-4: The CLCD window



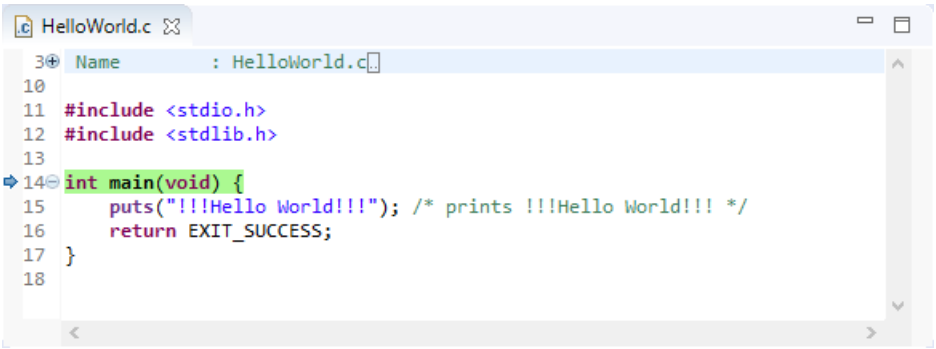
- Arm Development Studio connects to the model and displays the connection status in the **Debug Control** view.

Figure 6-5: Debug Control view



- The application loads on the target, and stops at the `main()` function, ready to run.

Figure 6-6: main () in code editor



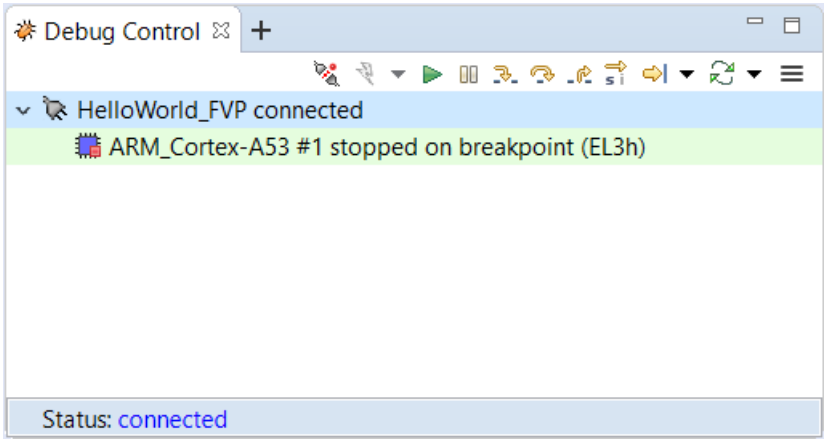
7. Application debug with Arm Debugger

Now that you have created a debug configuration and the application is loaded on the target, it is time to start debugging and stepping through your application.

Running and stepping through the application

Use the controls provided in the **Debug Control** view to debug your application. By default, these controls do source level stepping.

Figure 7-1: Debug Control view



The **Debug Control** view has the following controls:

 - Click to continue running the application after loading it on the target.

 - Click to interrupt or pause executing code.

 - Click to step through the code.

 - Click to step over a source line.

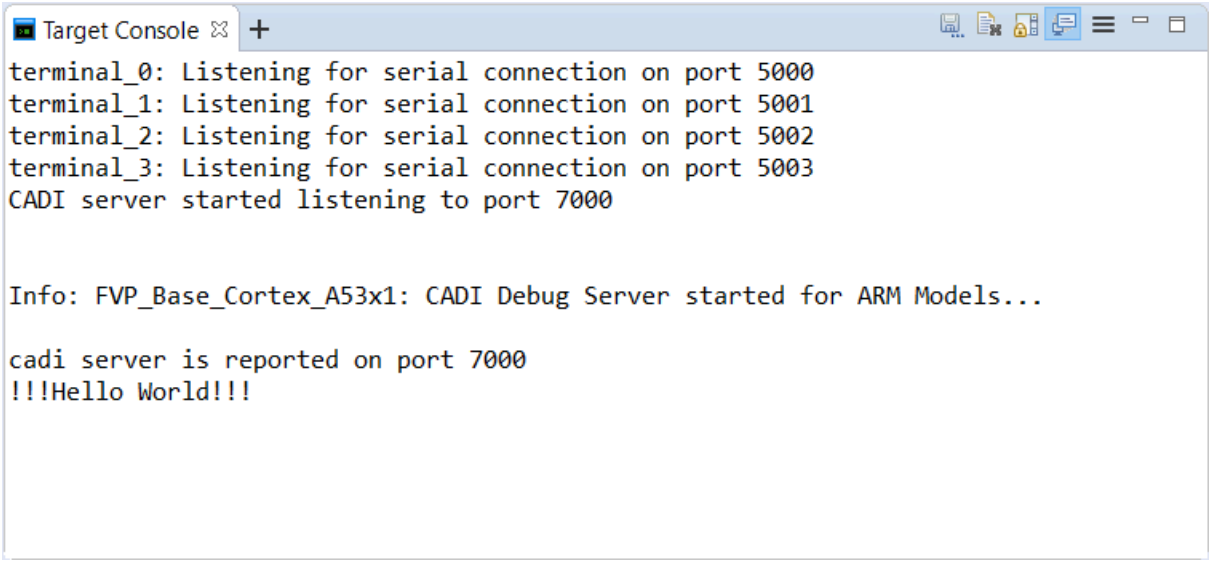
 - Click to step out.

 - This is a toggle. Click this to toggle between stepping instructions and stepping source code. This applies to the above step controls.

Other views display information relevant to the debug connection

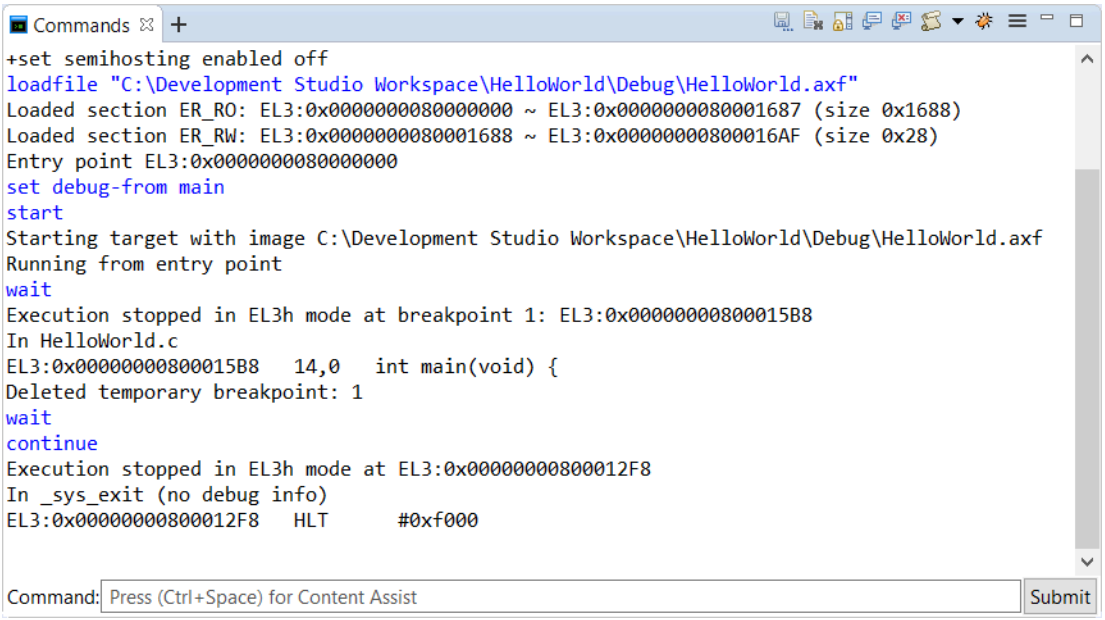
- **Target Console** view displays the application output.

Figure 7-2: Target console output



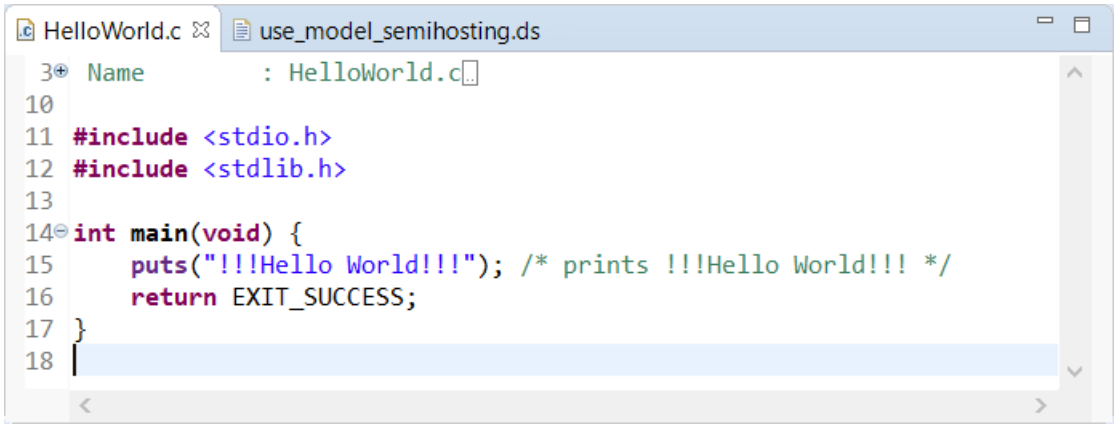
- **Commands** view displays messages output by the debugger. Also use this view to enter Arm® Debugger commands.

Figure 7-3: Commands view



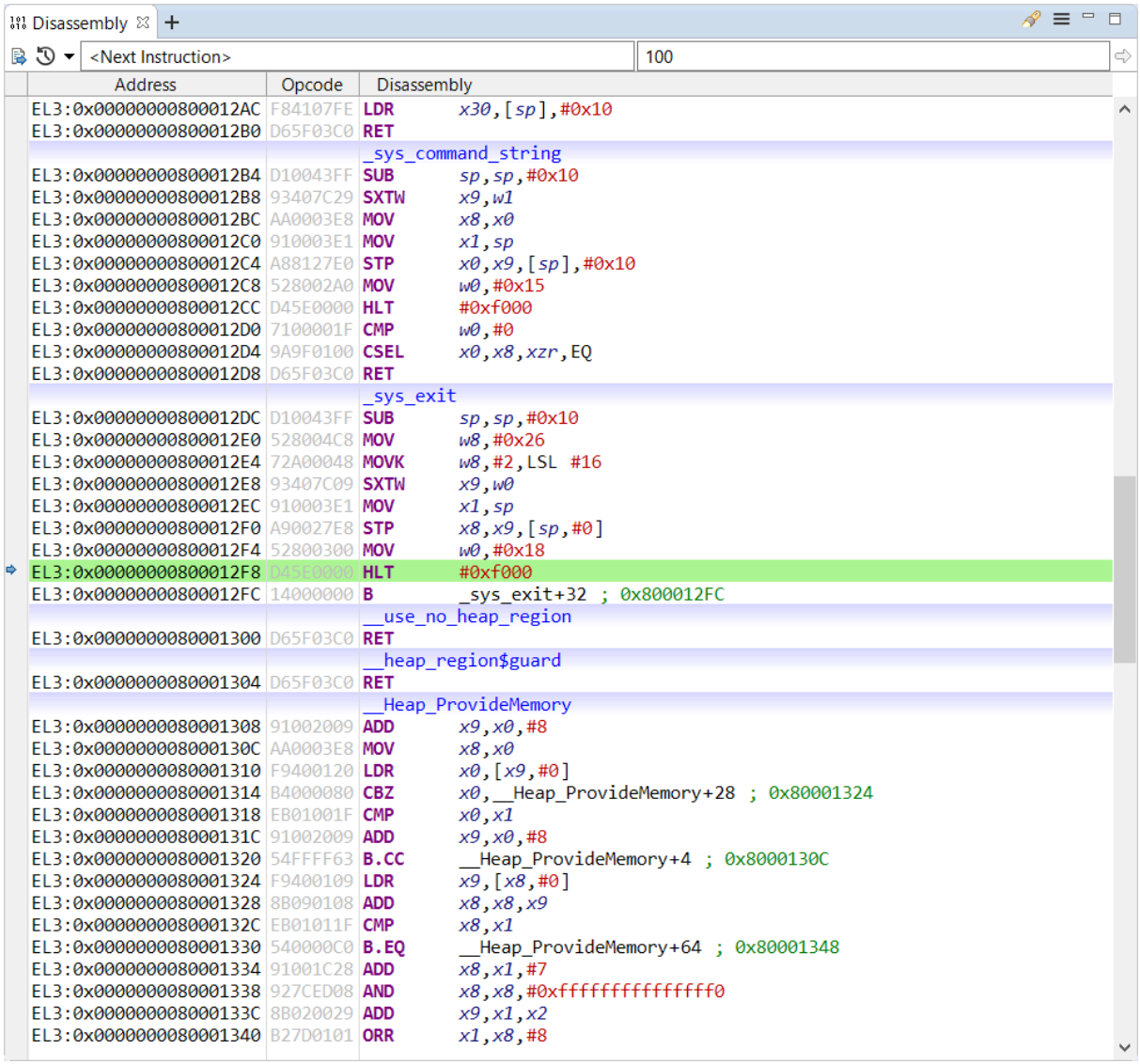
- C/C++ Editor view shows the active C, C++, or Makefile. The view updates when you edit these files.

Figure 7-4: Code Editor view



- **Disassembly** view shows the built program as assembly instructions, and their memory location.

Figure 7-5: Disassembly view

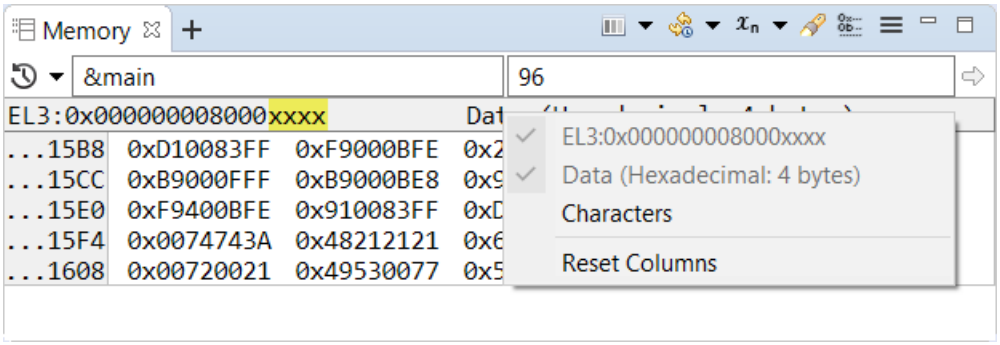


indicates the location in the code where your program is stopped. In this case, it is at the `main()` function.

- **Memory** view shows how the code is represented in the target memory. For example, to view how the string `Hello World` from the application is represented in memory:

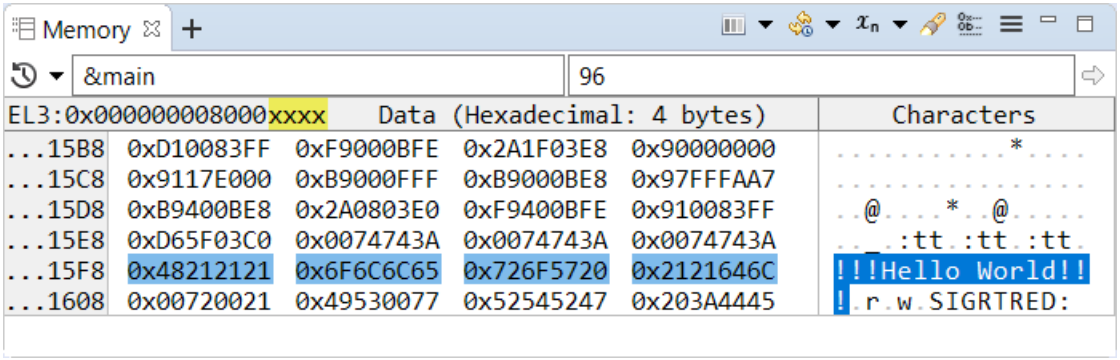
  1. Open the **Memory** view.
  2. In the **Address** field, enter `&main` and press **Enter** on your keyboard. The view displays the contents of the target's memory.
  3. Change the displayed number of bytes to `96` and press **Enter**.
  4. Right-click on the column headings, and select **Characters**.

Figure 7-6: Adding Characters column to Memory view



5. Select and highlight the words *Hello World*.

Figure 7-7: Memory view



In the above example, the **Memory** view displays the hexadecimal values for the code and the ASCII character encoding of the memory values, which enable you to view the details of the code.

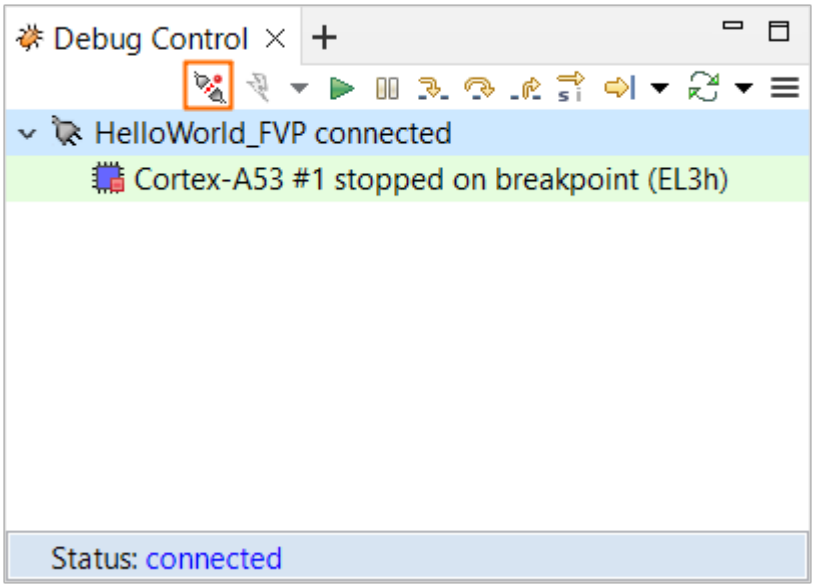
After completing your debug activities, you can Disconnect from the target.

8. Disconnect from the target

To disconnect from a target, you can use either the **Debug Control** or the **Commands** view.

- If you are using the **Debug Control** view, click **Disconnect from Target** on the toolbar.

  Figure 8-1: Disconnecting from a target using the Debug Control view



- If you are using the **Commands** view, enter **quit** in the **Command** field, then press **Enter**.

The disconnection process ensures that the state of the target does not change, except for the following case:

- Any downloads to the target are canceled and stopped.
- Any breakpoints are cleared on the target, but are maintained in Arm® Development Studio.
- The DAP (Debug Access Port) is powered down.
- Debug bits in the DSC (Debug Status Control) register are cleared.

If a trace capture session is in progress, trace data continues to be captured even after Arm Development Studio has disconnected from the target.

9. Capture trace output from an FVP

Trace capture from a Fixed Virtual Platform (FVP) provides you with a detailed output of all the instructions that are executed in a debug session. You can enable trace capture in the **Debug and Trace Services Layer (DTSL) Configuration** dialog box.

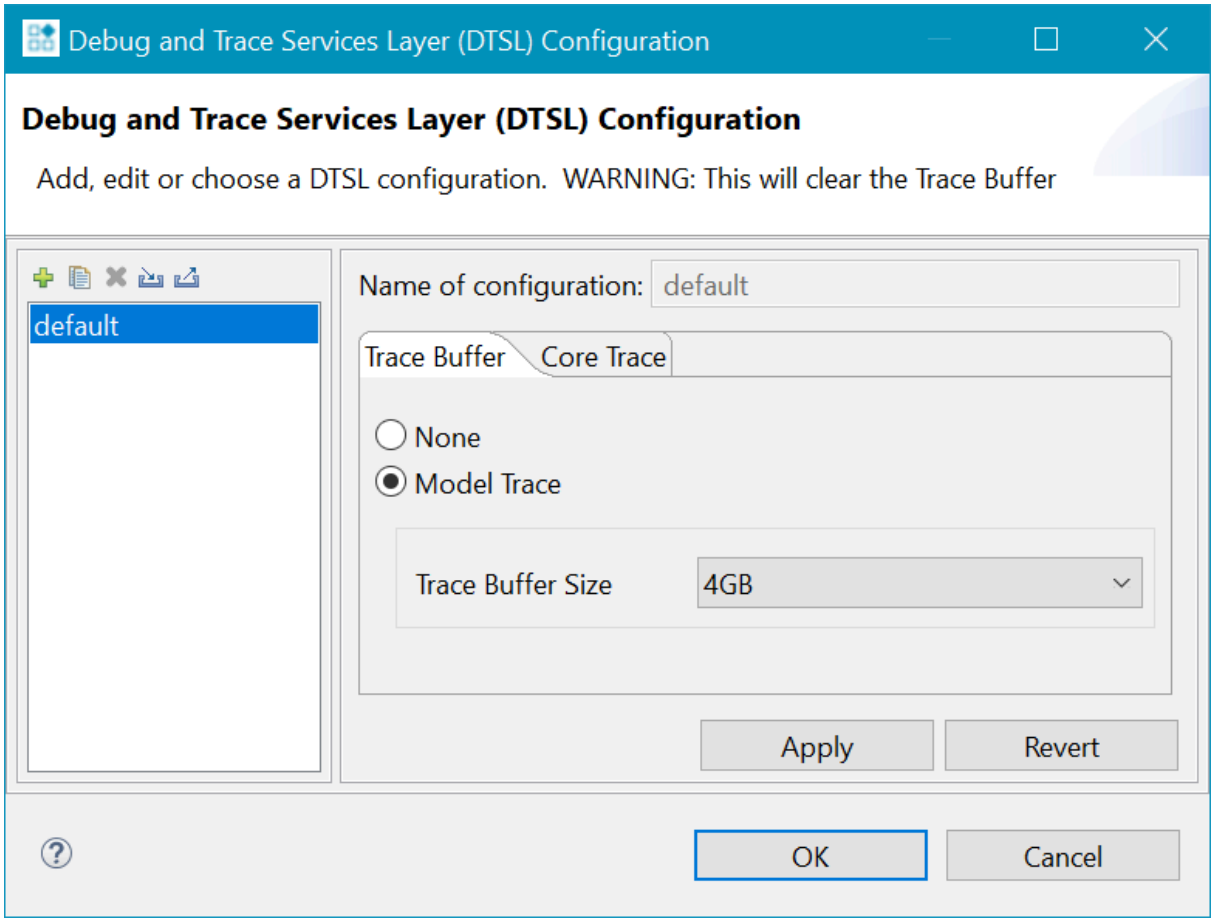
Procedure

1. In the **Debug Control** view, right-click on a disconnected target connection and select **DTSL Options**.
2. In the **Debug and Trace Services Layer (DTSL) Configuration** dialog box, select the **Model Trace** option under the **Trace Buffer** tab.

Note

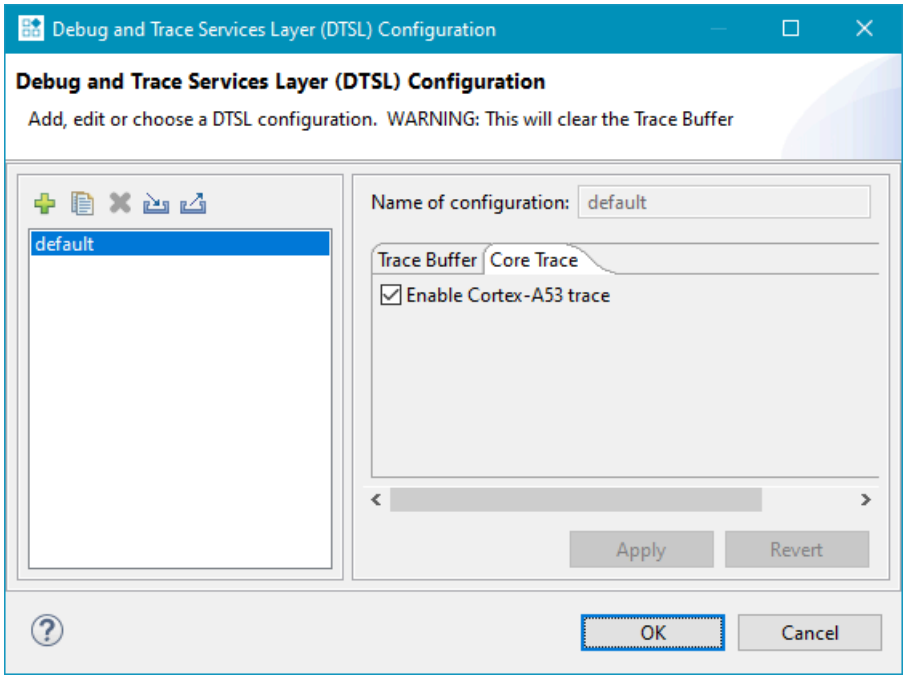
Here you can also change the trace buffer size in the **Trace Buffer Size** drop-down menu.

Figure 9-1: Trace Buffer tab



3. In the **Core Trace** tab, select the processor on which you want to enable trace capture.

Figure 9-2: Core Trace tab

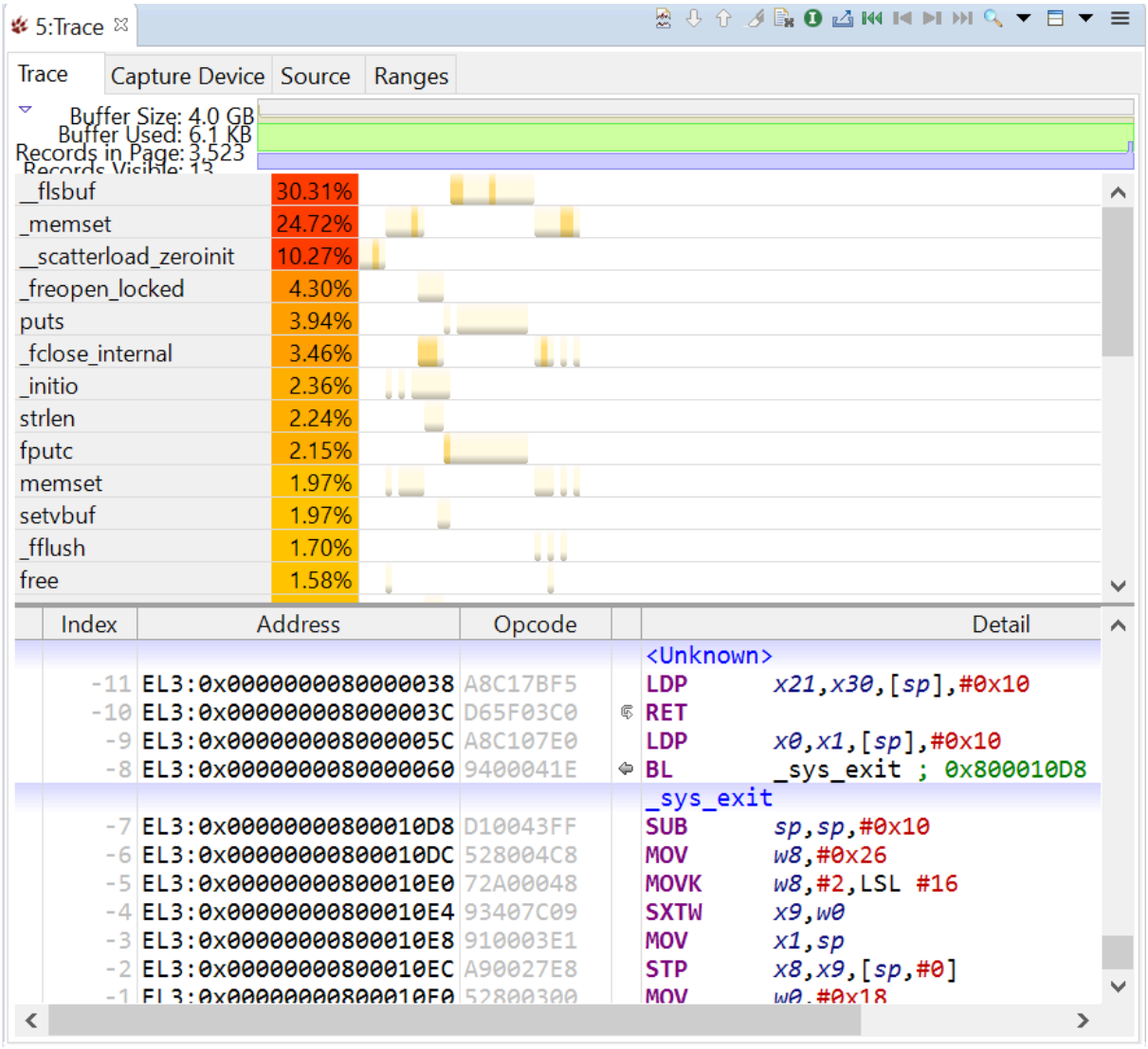


4. Select **Apply** and then **OK** to apply your settings and close the dialog box.

Next steps

1. Connect to the target.
2. In the **Trace** view, you can see all the instructions that are executed in a debug session.

Figure 9-3: Example trace capture in the Trace view



10. Other tutorials and workbooks

The following tutorials and workbooks might also be of interest:

- Beyond Hello World - advanced Arm Compiler 6 features

  This tutorial explores some of the more advanced features of the Arm® Compiler 6 toolchain.
- Arm Debugger Manual Configuration

  This workbook describes how to manually create a platform configuration for a specific target with Arm Development Studio *Platform Configuration Editor* (PCE). For the majority of targets, you can create a platform configuration automatically by performing target auto-detection with PCE. However, manually configuring a target can help you understand:
  - The information required to create a platform configuration
  - How a platform configuration is created
  - Which CoreSight™ devices are associated with debug and trace
  - How and why CoreSight devices are connected together
  - Important settings for the CoreSight devices
- Heterogeneous system debug with Arm Development Studio

  This workbook describes how to set up and debug the NXP i.MX7 SABRE board development board with Arm Development Studio. It takes you through the process of installing a Linux image, and then guides you through a debug session with bare-metal and Linux applications.
- Accessing memory-mapped peripherals with Arm Development Studio

  In most Arm embedded systems, peripherals are at specific addresses in memory. In your code, you must consider not only the size and address of the register, but also its alignment in memory. This tutorial describes how to map a C variable to each register of a memory-mapped peripheral and use a pointer to that variable to read from and write to the register.
- Debugging with the MCIMX8M-EVK board, DSTREAM-ST, and Arm Development Studio

  This tutorial describes 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 learn about the different features provided by Arm Development Studio including:
  - 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 access 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
- Tutorial: Hello World

This tutorial using the latest version of Arm Development Studio.