



Building bare-metal applications in DS-5 using GCC compiler

Version 1.0

Non-Confidential

Copyright © 2020 Arm Limited (or its affiliates).
All rights reserved.

Issue 02

102719_0100_02_en



Building bare-metal applications in DS-5 using GCC compiler

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	24 November 2020	Non-Confidential	Initial release

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, has undertaken no analysis to identify or understand the scope and content of, third party 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.

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 subsidiaries) 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 <https://www.arm.com/company/policies/trademarks>.

Copyright © 2020 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.

Feedback

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

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

Inclusive language commitment

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

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

Contents

1. Overview.....	6
2. Creating a new C project.....	7
3. Configuring the settings for the project.....	9
4. Modifying the GCC shared linker script.....	12
5. Creating the source code and building the project.....	14
6. Creating a DS-5 debug configuration and connecting to the FVP.....	15

1. Overview

This tutorial shows you how to set up your project to use the GCC bare-metal compiler. It then guides you through creating a simple bare-metal Hello World application and finally running it on a debug configuration on a Cortex-A9 Fixed Virtual Platform (FVP) provided with DS-5.

Including set-up for bare-metal debug sessions on Fixed Virtual Platforms

To debug applications for bare-metal targets in Arm DS-5 Development Studio, you can use GCC “Launchpad” compiler toolchain. However, in general we recommend use of Arm Compiler for building bare-metal applications in DS-5.

This tutorial shows you how to set up your project to use the GCC bare-metal compiler. It then guides you through creating a simple bare-metal Hello World application and finally running it on a debug configuration on a Cortex-A9 Fixed Virtual Platform (FVP) provided with DS-5.

Prerequisites

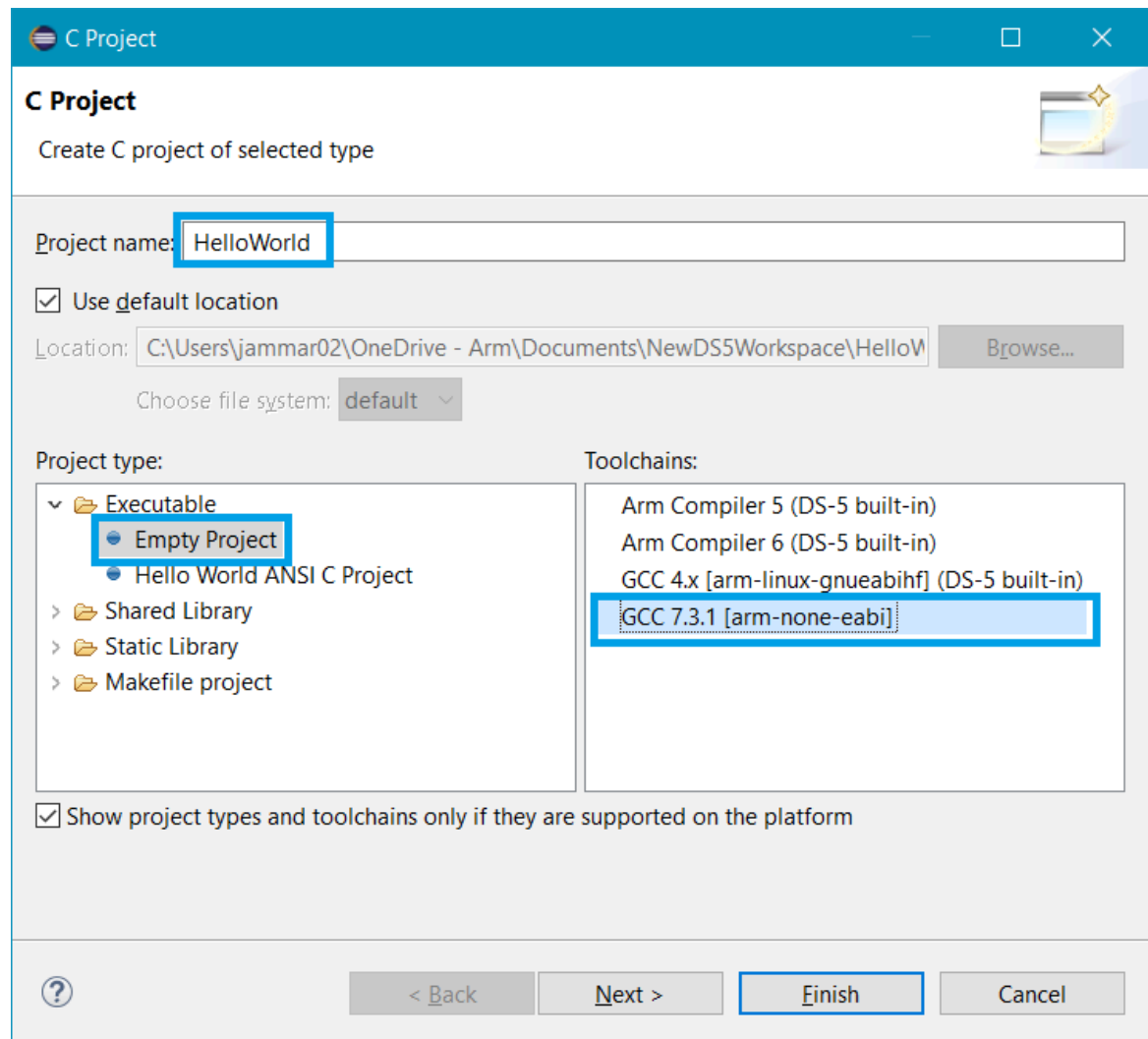
- Download, install, and acquire a license for DS-5. If you haven't, see the [Getting Started with Arm DS-5](#) tutorial for more information.
- Download and install the GCC bare-metal toolchain:
 - If you are compiling for Cortex-A, select a toolchain from linaro.org.
 - For more information regarding the Linaro Toolchain releases, support, and selection, go to <https://wiki-archive.linaro.org/WorkingGroups/ToolChain/FAQ>.
 - If you are compiling for Cortex-R or Cortex-M, select a toolchain from [GNU Arm Embedded Toolchain](#).
 - For more information regarding the GNU Arm Embedded Toolchain releases, support, and selection, go to [GNU Arm Embedded Toolchain](#).
 - If you need help adding a new toolchain to Arm DS-5, see the tutorial on [Adding New Compiler Toolchains to DS-5](#).

2. Creating a new C project

To create a new C project follow the steps:

1. From the DS-5 main menu, select File > New > C Project to display the C Project dialog.
2. In the C Project dialog:
 - a. In the Project name field, enter HelloWorld as the name of your project.
 - b. Under Project type, select Executable > Empty Project.

Figure 2-1: C Project dialog options for GCC



When selecting a Bare-metal toolchain option, the toolchain assumes that the application is executed directly on the hardware instead of on top of a complex operating system such as Linux.

- c. Under Toolchains, select the name of the GCC toolchain you have downloaded.

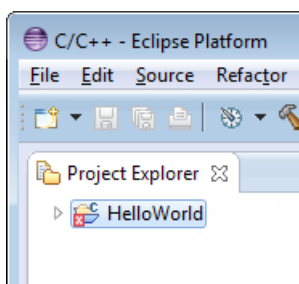


If the toolchain that you downloaded is not listed, you need to [add it to the list](#).

- d. Click Finish to create a C project called HelloWorld.

You can view the project in the Project Explorer view.

Figure 2-2: Project Explorer view



You might see a red cross on the project which indicates that the project configuration is incomplete, depending on your version of DS-5. The next sections explain how to set up the rest of the settings for the project.

3. Configuring the settings for the project

You need to be aware that when configuring settings for a project using a compiler not shipped with DS-5 (but supported by DS-5), there might be options and settings that you will have to investigate before applying any changes. For example, you need to know the processor you are compiling your code for and you might also need to investigate the RAM starting address for your target, so you can specify it in the linker script.

For this tutorial, we are going to customize the project to:

- Set up the appropriate commands and flags for the GCC C Compiler, GCC Assembler, and GCC C Linker. This tells the project to use specific switches when compiling the program.
- Configure the project environment to locate and use the GCC compiler executable.
- Modify the GCC linker script to set the RAM starting address.

Setting up the commands and flags for the GCC C Compiler, GCC Assembler, and GCC C Linker

Once you have created your project, you need to specify the commands and flags required to compile it.



This configuration exists on a per-project basis. You must separately reconfigure all projects that you want to use with the new toolchain.

Setting up project commands and flags:

1. Locate your project in the Project Explorer view, right-click it, and select Properties.
2. In the Properties dialog:
 - a. Navigate to C/C++ Build > Settings.
 - b. Select All configurations in the Configuration list.
 - c. Select GCC C Compiler, and confirm that you are using an arm-none-eabi-gcc toolchain, then check if the command is available under Commands. Similarly, confirm the command is available under GCC Assembler, and GCC C Linker.
 - d. Select Target and in the CPU (-mcpu) field, enter cortex-a9.
 - e. Select GCC C Linker > Libraries, and:
 1. Click the add library button to Add a library, and in the Enter Value dialog, enter `c`.
 2. Click the add library button to Add a library again, and in the Enter Value dialog, enter

```
rdimon  
enables semihosting on your target.
```

Semihosting is a mechanism that enables code running on an ARM target to communicate and use the Input/Output facilities on a host computer that is running a debugger.

For more information about semihosting, see [What is semihosting?](#)

3. Select GCC C Linker > Miscellaneous and in the Other flags field, enter:

```
--specs=nano.specs --specs=rdimon.specs -mcpu=cortex-a9 -T ../gcc.ld
```

The `gcc.ld` is the shared linker script that we will add to the project and modify [at a later step](#).

The commands and flags for the GCC compiler to compile code using the appropriate libraries and processor switches are now set up. Now, you need to set up the project environment variable to point to the GCC compiler executable which is installed separately from DS-5. Leave the project Properties dialog open to configure the project Environment settings.

Check out this blog for some information about GCC command line options for [Arm Cortex-A processors: Arm Cortex-A Processors and GCC Command Lines](#)

Set up the project environment to use the GCC compiler executable

To ensure DS-5 can find the GCC compiler executable, add its location to the `PATH` environment variable.

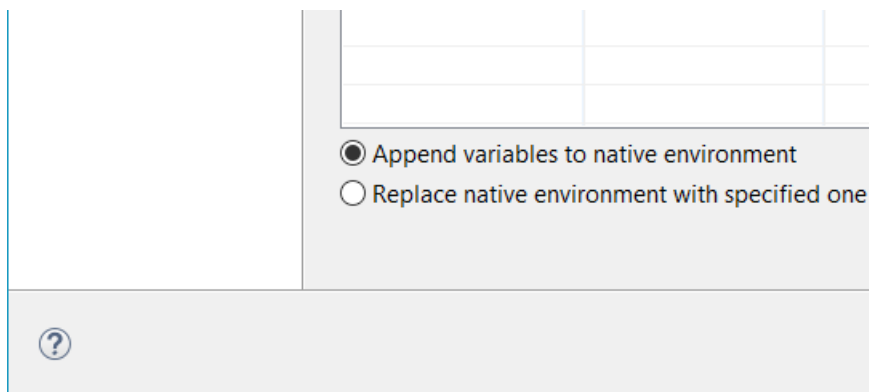


This configuration exists on a per-project basis. You must separately reconfigure all projects that you want to use with the new toolchain.

In the project Properties dialog:

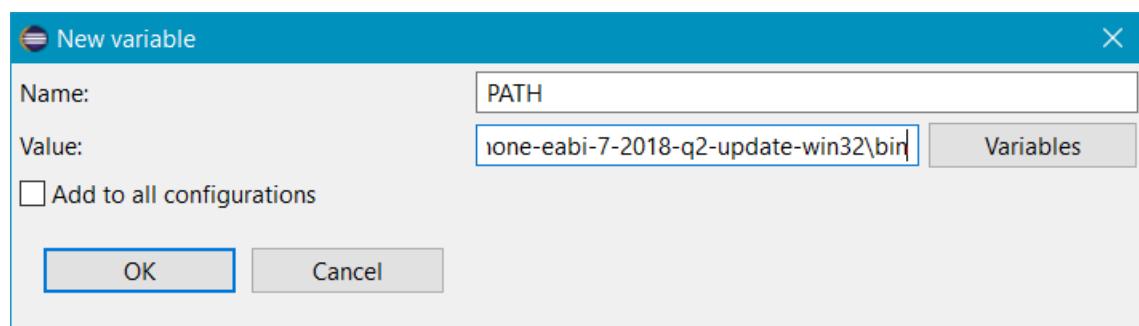
1. Navigate to C/C++ Build > Environment.
2. Ensure the Append variables to native environment option is selected.

Figure 3-1: Append Variables option.



3. Click Add.
4. In the New variable dialog:
 - a. In the Name field, enter PATH.
 - b. In the Value field, enter the path where the GCC compiler is installed. For example: `c:\Program Files (x86)\GNU Tools ARM Embedded\4.7 2013q3\bin`

Figure 3-2: New PATH variable dialog.



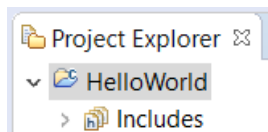
- c. Click OK to save the changes and close the Edit variable dialog.
5. Click OK to save the changes and close the project Properties dialog.

4. Modifying the GCC shared linker script

After setting up the new project with appropriate flags, you now need to copy and modify the GCC shared linker script so that the linker knows what the RAM starting address for the FVP is.

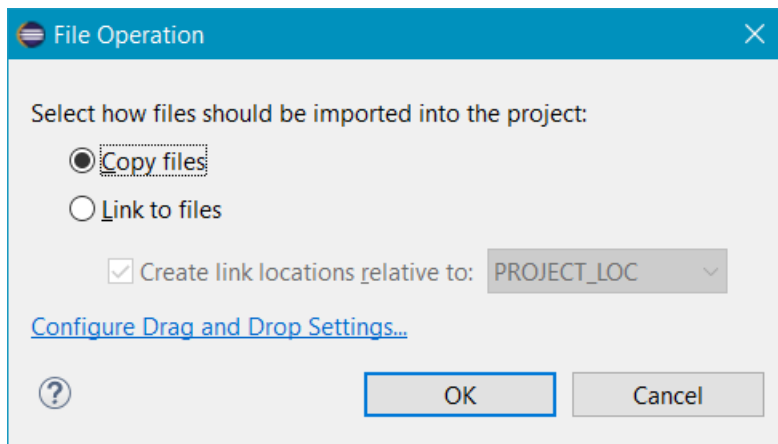
1. Locate the `gcc.ld` file in: `C:\Program Files (x86)\GNU Tools ARM Embedded\4.7 2013q3\share\gcc-arm-none-eabi\samples\ldscripts`
2. Drag and drop the file from the location on your computer to the project in DS-5.

Figure 4-1: Drag and Drop gcc.ld file.



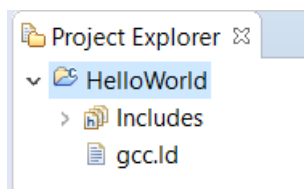
3. In the File Operation dialog, select Copy files and click OK to copy the file and close the dialog.

Figure 4-2: File Operation dialog.



4. Expand the project to view the copied file.

Figure 4-3: gcc.ld file in project.



5. Double-click to open the `gcc.ld` file in the default editor set up in DS-5.
6. Change:

```
MEMORY
{
    FLASH (rx) : ORIGIN = 0x0, LENGTH = 0x20000 /* 128K */
}
```

```
RAM (rwx) : ORIGIN = 0x10000000, LENGTH = 0x2000 /* 8K */  
}
```

To

```
MEMORY  
{  
  RAM (rwx) : ORIGIN = 0x80000000, LENGTH = 0x2000 /* 8K */  
}
```

Where can I locate more information about Versatile Express FVPs and their memory maps?

The [Fixed Virtual Platforms FVP Reference Guide](#) contains more information. See [VE - model memory map](#) for information specific to VE memory maps.

7. Locate `ENTRY(Reset_Handler)` and change it to `ENTRY(_start)`.
8. Locate all instances of `} > FLASH` and change it to `} > RAM`.
9. Save the file.

After creating the shared linker script, you need to create a target initialization debugger script.

5. Creating the source code and building the project

Now that the project is set up, we can start creating code for it and build the application. After a successful build, we can then create a debug configuration, and run the application on the target.

1. In the Project Explorer view, right-click the HelloWorld project and select New > Source File.
2. In the New Source File dialog, enter the file name `hello_world.c`.
3. Click Finish to create the source file and open it in the code editing view. The source file is also visible in the Project Explorer view, under the Hello World project.
4. Add the following code to the new source file, and press CTRL+S to save it.

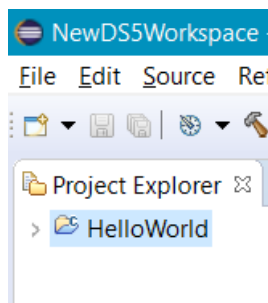
```
#include <stdio.h>
int main(int argc, char** argv)
{
    printf("Hello world\n");
    return 0
}
```

5. In the Project Explorer view, right-click on the Hello World project and select Build Project.

You can view the output image `hello_world.axf` in the Debug folder under the HelloWorld project.

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

Figure 5-1: Compiled GCC project.

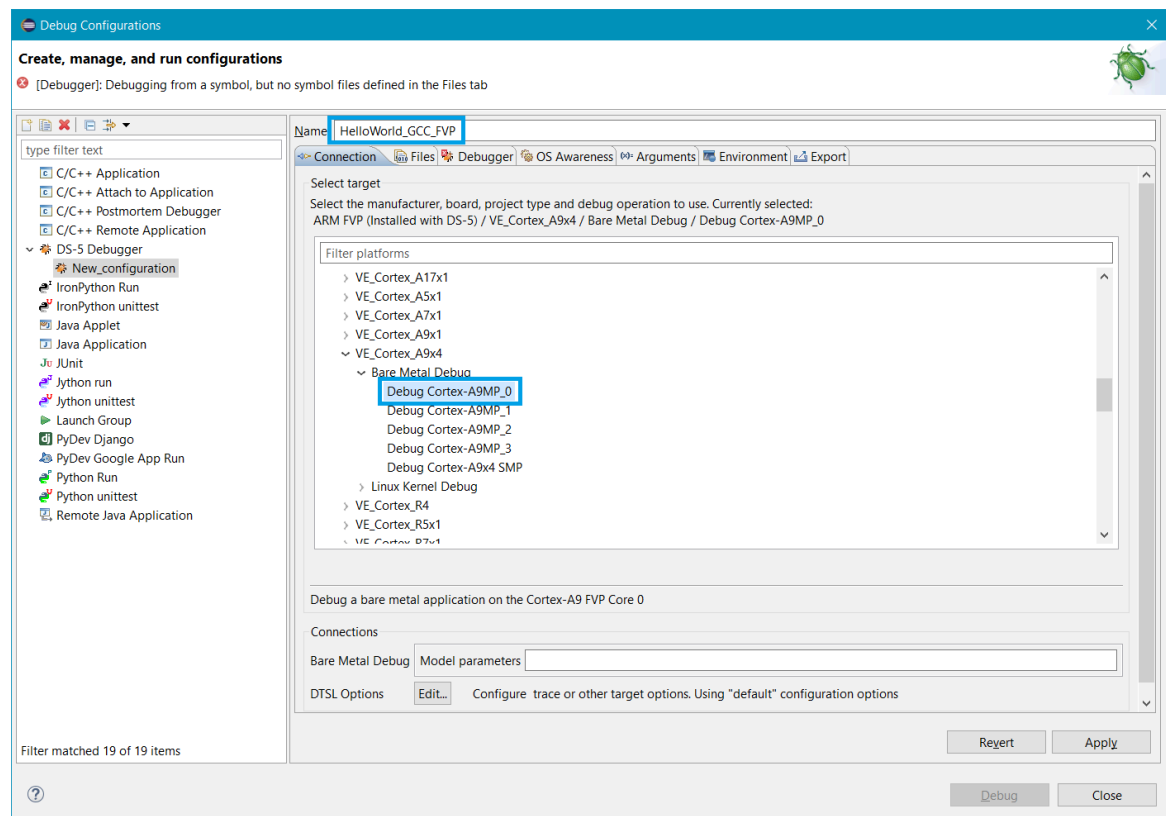


6. Creating a DS-5 debug configuration and connecting to the FVP

To create a DS-5 debug configuration and connecting to the FVP follow the steps:

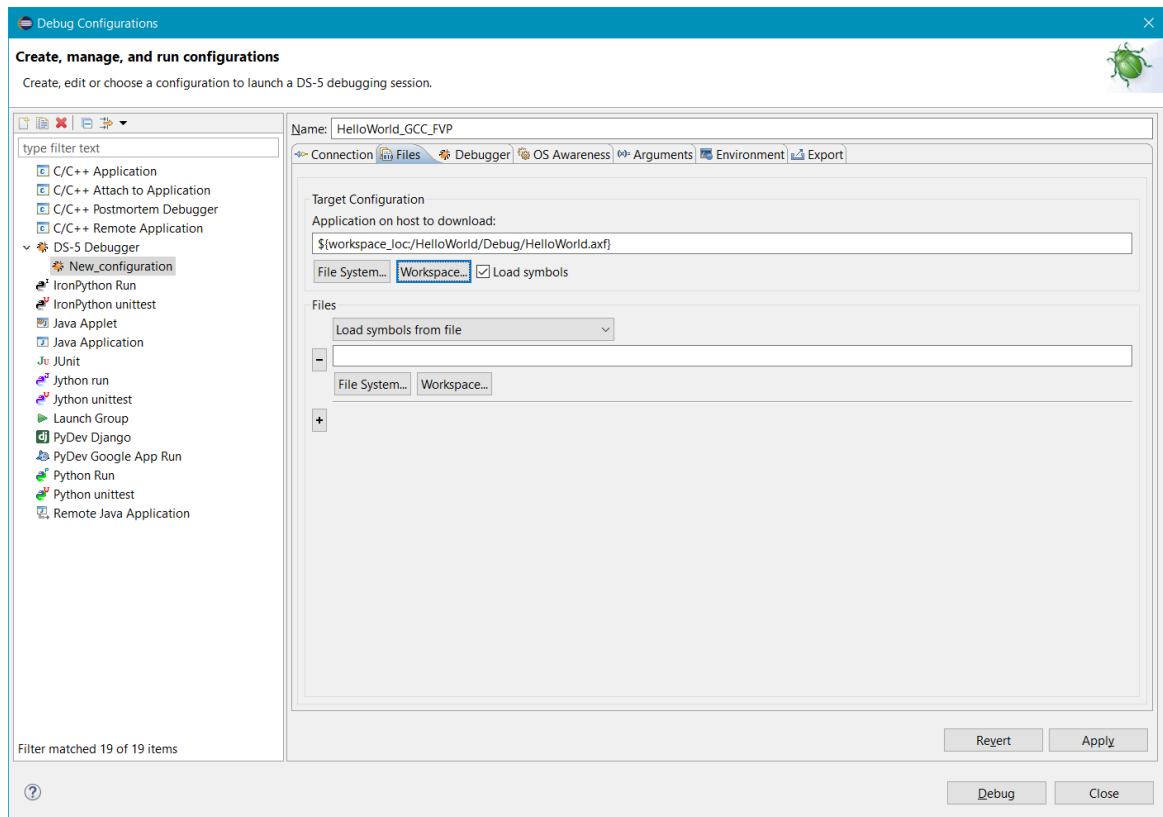
1. From the DS-5 main menu, select Run > Debug Configurations.
2. In the Debug Configurations dialog:
 - a. Select DS-5 Debugger.
 - b. Click the New launch configurations button.

Figure 6-1: Debug configurations - New



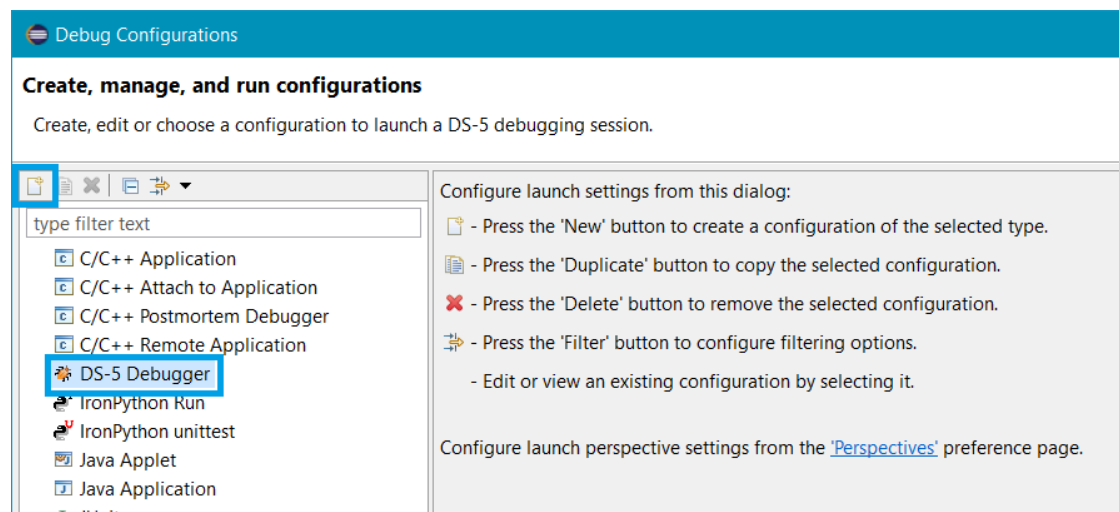
This creates a new DS-5 debug configuration and displays the various tabs required to specify settings for loading your application on the target.

Figure 6-2: Debug configurations - Tabs



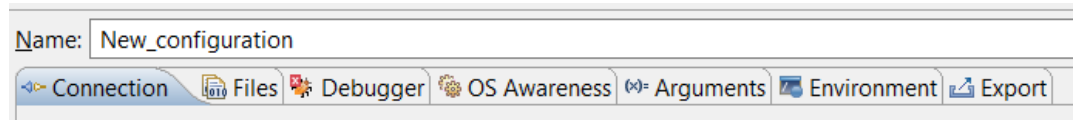
- c. In the Debug Configurations dialog:
 1. Give a name to the debug configuration. For example, HelloWorld_GCC_FVP.
 2. In the Connection tab, select Arm FVP > VE_Cortex_A9x4 > Bare-Metal Debug > Debug Cortex-A9_0.

Figure 6-3: Debug configurations - Debugger tab



3. Select the Files tab, and:
 - a. Under Target Configuration in the Application on host to download field, click Workspace.

Figure 6-4: Debug configurations - Files Tabs



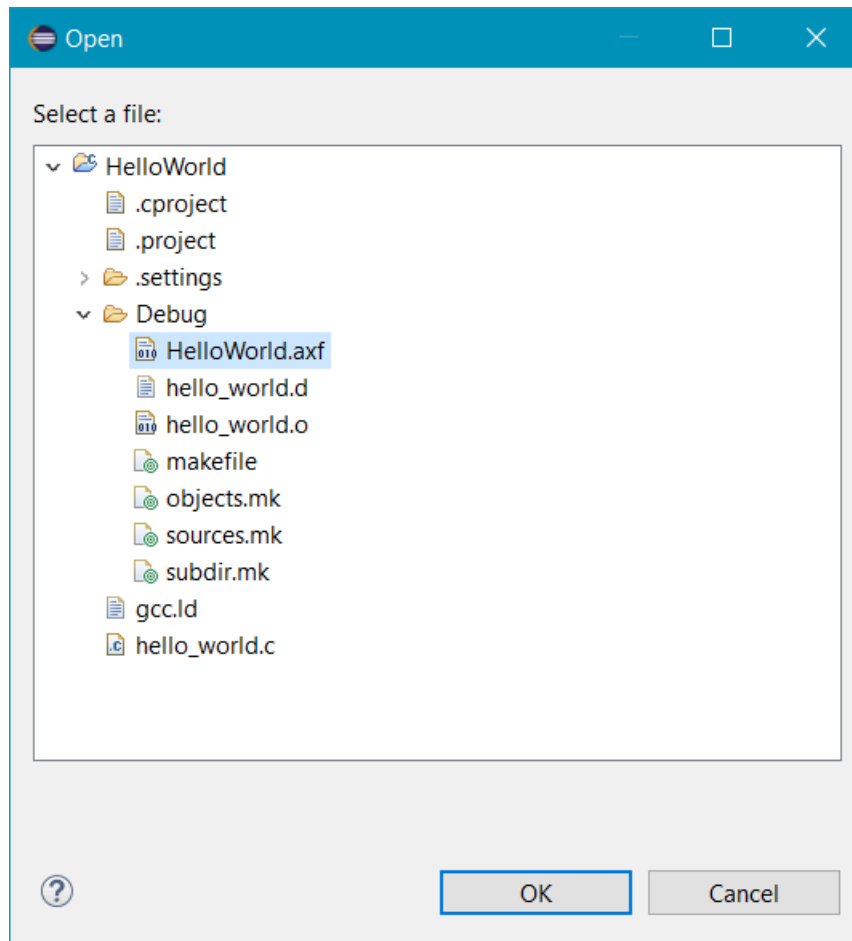
The Workspace contains the HelloWorld.axf application file you created when you built the Hello World project.



Ensure that the Load symbols option is selected.

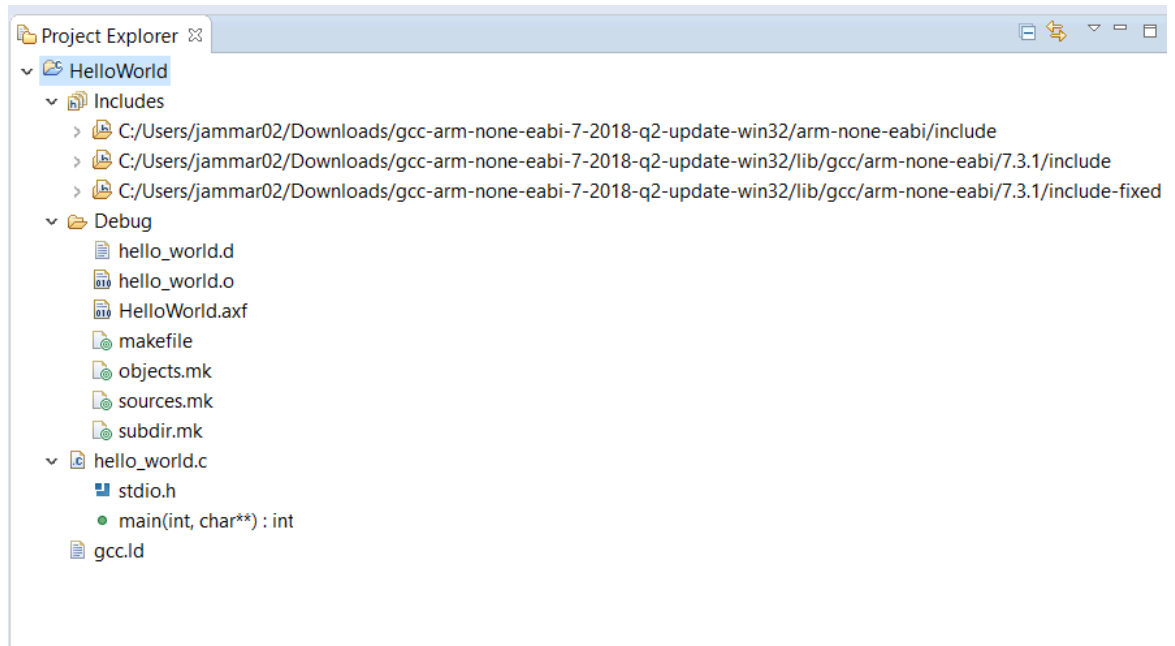
- b. Select HelloWorld.axf.

Figure 6-5: Open Hello World



- c. Click OK to load the file.
- d. Select the Debugger tab, and ensure the Debug from symbol option is selected and set to main.
- e. Click Debug to load the application on the FVP, and load the debug information into the debugger.
- f. In the Confirm Perspective Switch dialog that appears, click Yes. DS-5 connects to the FVP and displays the connection status in the Debug Control view.

Figure 6-6: Debug Control view - Altera GCC configuration



The application is loaded on the target, and has stopped at the `main()` function, ready to run.

- g. Click the continue button to continue running the application. You can view the application output in the **Target Console** view.

Figure 6-7: Target Console output

