# arm

# Debugging on bare-metal targets using DS-5 and GCC compiler

Version 1.0

# Debugging on bare-metal targets using DS-5 and GCC compiler

## Release information

**Document history**

| Issue | Date | Confidentiality | Change |
|-------|------|-----------------|--------|
| 0100-02 | 1 January 2020 | Non-Confidential | Initial release |

## Proprietary Notice

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.

## 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

To debug applications for bare-metal targets in Arm DS-5 Development Studio, you can use GCC compiler. This is useful in situations where you do not have access to Arm Compiler, for example, when using an edition of DS-5 that is licensed for a specific toolkit.

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 for the Altera® Cyclone® V SoC.

## Prerequisites

* Download, install, and acquire a license for DS-5. If you have not, see the Getting Started with Arm DS-5 tutorial for more information.

* Download and install the GCC bare-metal toolchain. You can download this from: linaro.org/downloads

* Ensure that your target is powered on and is working.

* This tutorial uses USB-Blaster as the target connection type. To enable the connection, ensure that the environment variable `QUARTUS_ROOTDIR` contains the path to the Altera Quartus tools installation. On Windows, this environment variable is usually set by the Quartus tools installer. On Linux, you might have to manually set the environment variable to the Altera Quartus tools installation path. For example, `~/altera/13.0/qprogrammer`. For information on installing device drivers for USB-Blaster and USB-Blaster II, consult your Altera Quartus tools documentation.

# 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 > Project and in the New Project dialog, select C Project and click Next.

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: Screenshot of the empty project executable screen**
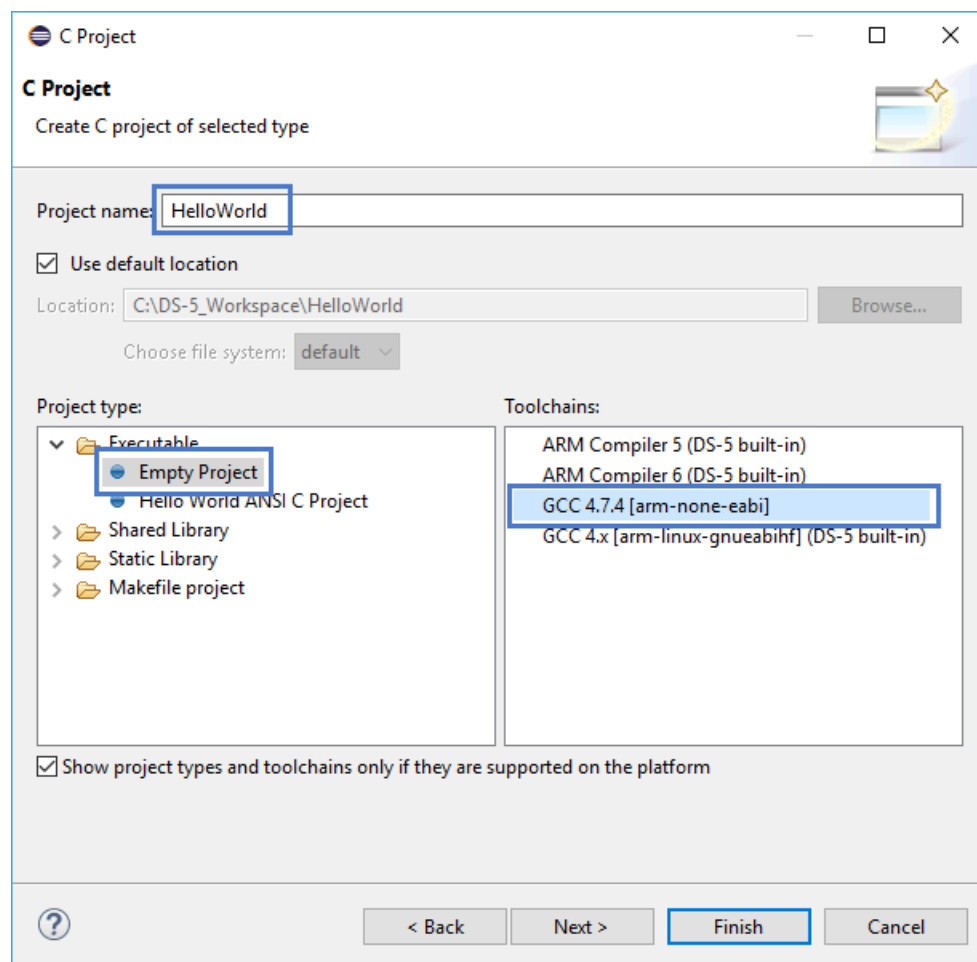


    c.  Under Toolchains, select your relevant GCC compiler (In this example, GCC 4.7.4 [arm-none-eabi]).

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

    e.  If the Open Associated Perspective dialog pops up, click Yes to switch to the relevant perspective.

You can view the project in the Project Explorer view.

**Figure 2-2: GCC project in Project Explorer**
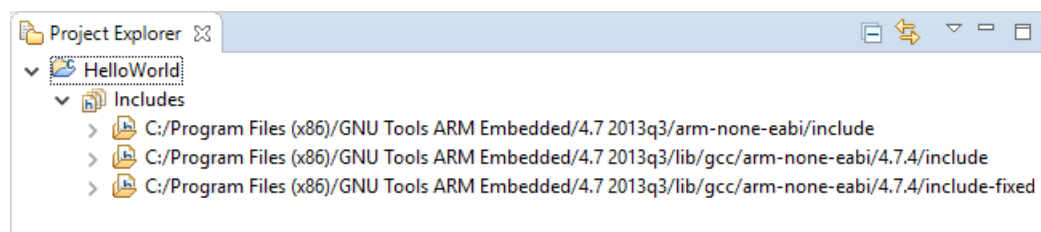


You have now created the project, but the project configuration is incomplete. The sections below explain how to set up the rest of the settings for the project.

# 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 which processor is available on your board, so you can compile your code for a specific processor. You might also need to investigate the RAM starting address for a particular board, 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.

- Create a target initialization debugger script to enable semihosting.

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

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

---

**Note**

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:

        1. Confirm if the `arm-none-eabi-gcc` command is available under Command. Similarly, confirm the command is available under GCC Assembler (`arm-none-eabi-as`), and GCC C Linker (`arm-none-eabi-gcc`).

        2. Select Target and in the CPU (`-mcpu`) field, enter `cortex-a9`.

    d. Select GCC C Linker > Libraries, and:

        1. Click 📋 to Add a library, and in the Enter Value dialog, enter c.

        2. Click 📋 to Add a library again, and in the Enter Value dialog, enter `rdimon`. `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?

e. 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.

---

**Note**

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**



◉ Append variables to native environment
○ Replace native environment with specified one
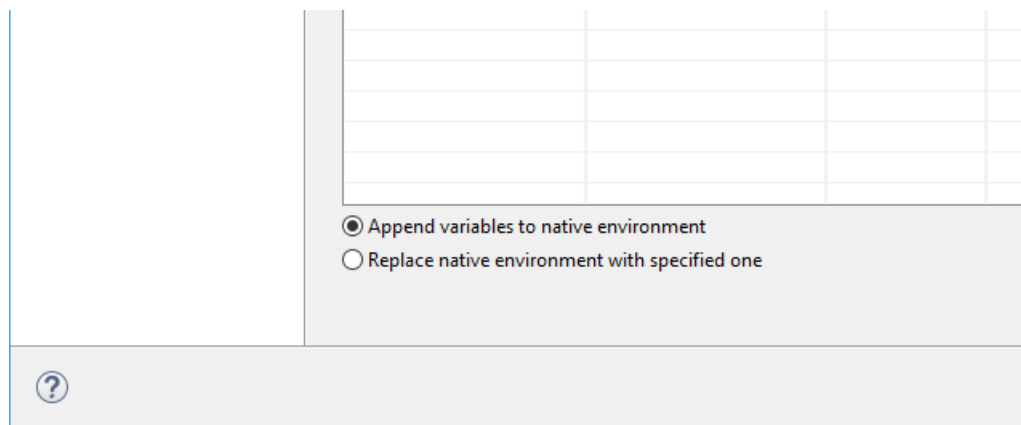
3. Click Add.

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**
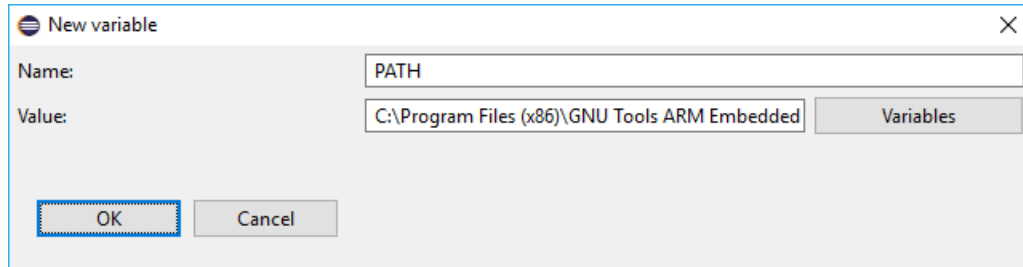


c. Click OK to save the changes and close the Edit variable dialog.

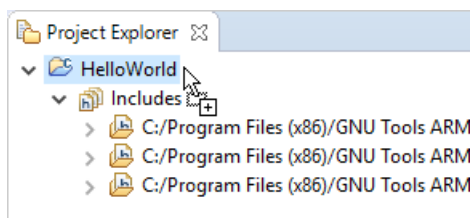5. Click OK to save the changes and close the project Properties dialog.

# 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 Altera Cyclone Board 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**
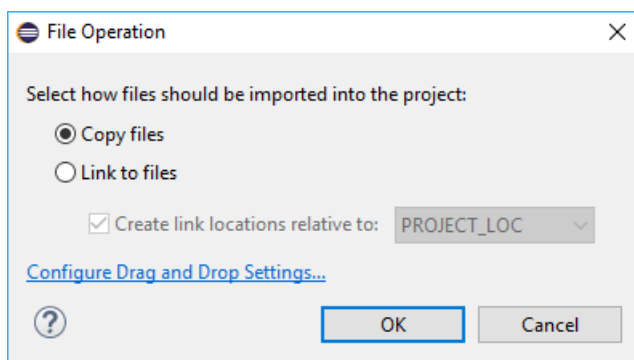
   

4. Expand the project to view the copied file.

   **Figure 4-3: View the copied file**
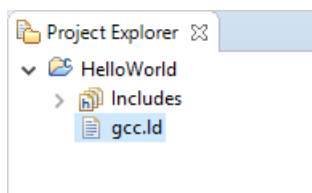
   

5. Double-click to open the `gcc.ld` file in the default editor set up in DS-5.

6. Change:

```
MEMORY
```

```
{
  FLASH (rx) : ORIGIN = 0x0, LENGTH = 0x20000 /* 128K */
  RAM (rwx) : ORIGIN = 0x10000000, LENGTH = 0x2000 /* 8K */
}
```

To

```
MEMORY
{
  RAM (rwx) : ORIGIN = 0xFFFF0000, LENGTH = 0x2000 /* 8K */
}
```

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 a target initialization debugger script

A target initialization script executes debugger commands on the target immediately after connection. For this tutorial, we are going to create a script which specifies the RAM starting address and the semihosting options.

1. From the DS-5 main menu, select File > New > Other.

2. In the New/Select a wizard dialog, select General > Untitled Text File and click Finish. This opens an empty text file in the default DS-5 editor.

3. Enter:

```
reset
pause 2000
set semihosting top-of-memory 0xFFFF2000
set semihosting enabled on
```

See the DS-5 Debugger Command Reference guide for additional set commands and for commands and options available for semihosting, for example, show semihosting, info semihosting, or set semihosting.

4. Press CTRL + S to Save the file.

5. In the Save As dialog, select the project to save the file to, which in this case is HelloWorld.

6. Click OK to close the Save As dialog. Expand the project to view the copied file.

**Figure 5-1: semihosting.ds file in project**
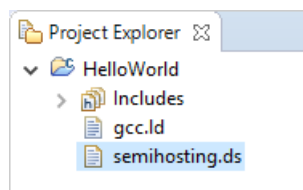
# 6. Creating the source code and building the project

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.

**Figure 6-1: HelloWorld.axf**
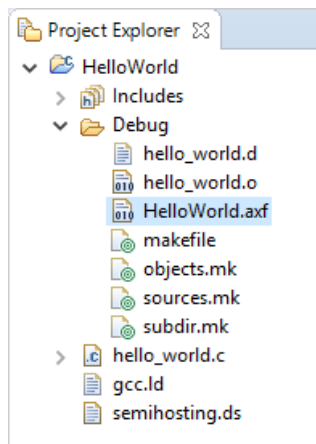


The `.axf` file contains both the object code and debug symbols that enable the debugger to perform source-level debugging.

Debugging on bare-metal targets using DS-5 and GCC
compiler

Document ID: 102609_0100_02_en
Version 1.0
Creating a DS-5 debug configuration and connecting to your Altera
target

# 7. Creating a DS-5 debug configuration and connecting to your Altera target

To create a DS-5 debug configuration and to connect to your Altera target, 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.
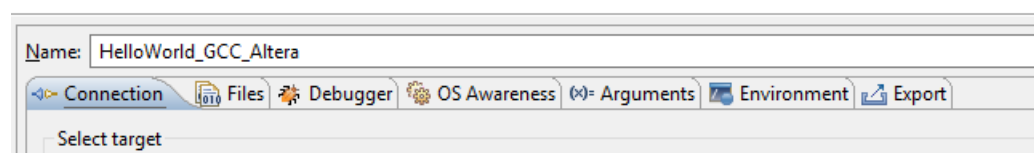   b. Click the New launch configurations button.

**Figure 7-1: Debug configurations**
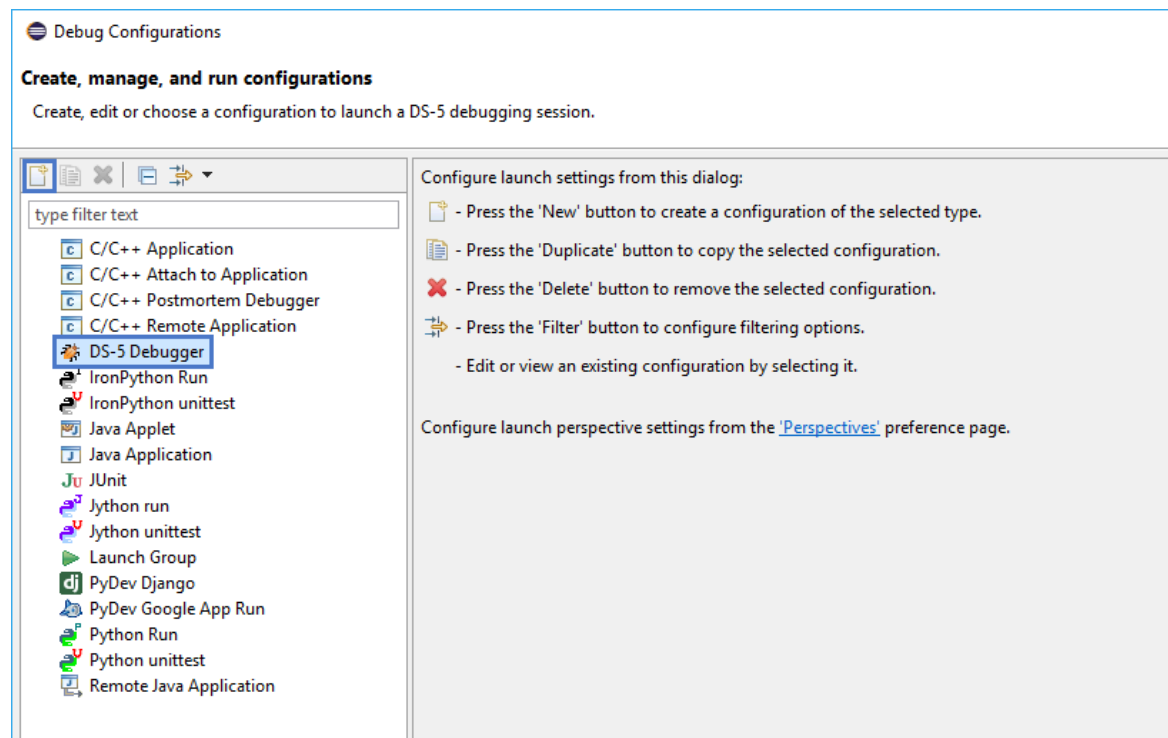


This creates a new DS-5 debug configuration and displays the various tabs required to specify settings for loading your application on the target.

   c. Give a name to the debug configuration. For example, `HelloWorld_GCC_Altera`.

**Figure 7-2: Name the debug connection**

Debugging on bare-metal targets using DS-5 and GCC
compiler

Document ID: 102609_0100_02_en
Version 1.0
Creating a DS-5 debug configuration and connecting to your Altera
target

3. In the Connection tab:

   a. Select Altera > Cyclone V SoC > Bare-Metal Debug > Debug Cortex-A9_0.

   b. In the Target Connection, select USB-Blaster.

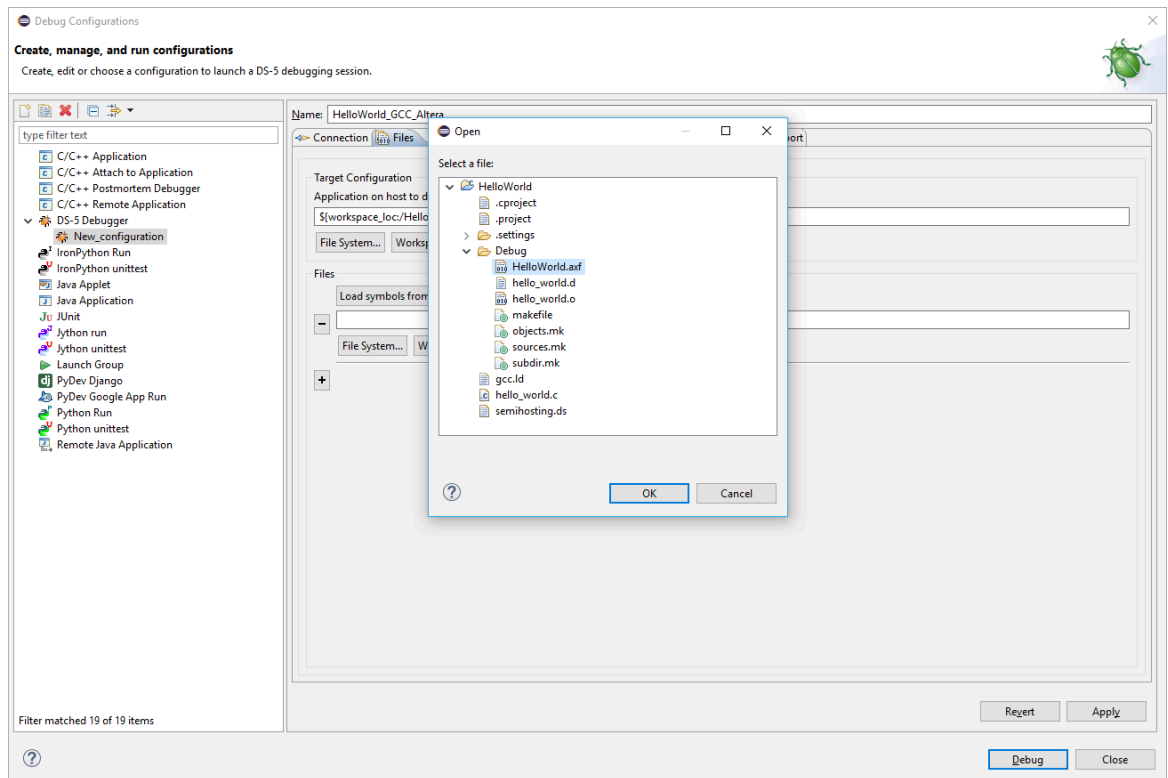   c. In the Bare Metal Debug/Connection area, click Browse and select your target.

   **Figure 7-3: Connection tab**



---


**Note**
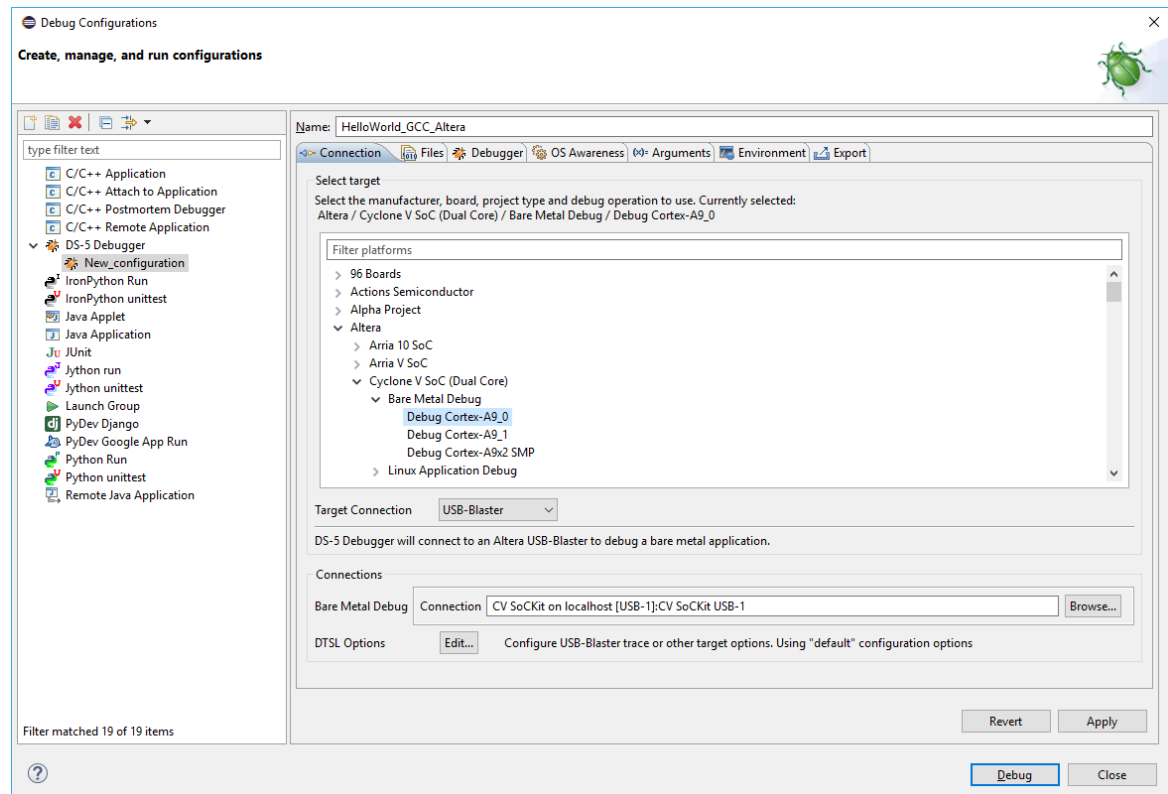
   Ensure your target is powered on before you attempt a connection to it.

---

4. Select the Files tab, and:

   a. Under Target Configuration in the Application on host to download field, click Workspace. The Workspace contains the `HelloWorld.axf` application file you created when you built the Hello World project.

   b. Select `HelloWorld.axf`.

Debugging on bare-metal targets using DS-5 and GCC
compiler

Document ID: 102609_0100_02_en
Version 1.0
Creating a DS-5 debug configuration and connecting to your Altera
target

**Figure 7-4: Debug configuration files tab**
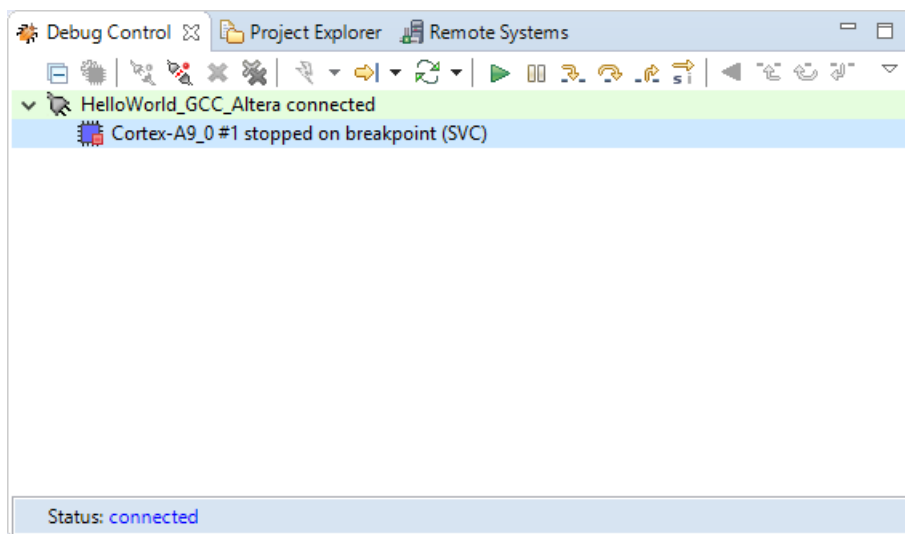


    c.   Click OK to load the file.

5.   Select the Debugger tab:

    a.   Ensure the Debug from symbol option is selected and set to main.

    b.   Select the Run target initialization debugger script (.ds / .py) option to enable it.

    c.   Click Workspace and browse and select the `.ds` target initialization script you created
        earlier.

**Figure 7-5: Debug configuration debugger tab**
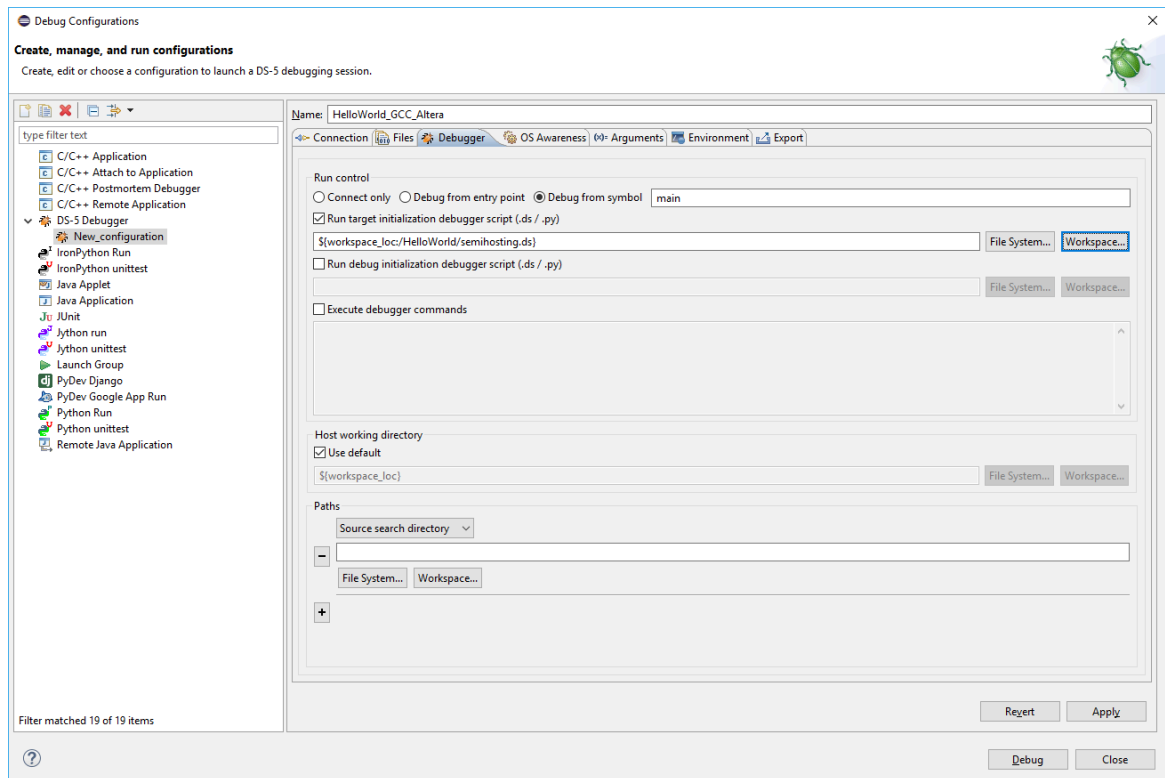


6. Click Debug to load the application on the target, and load the debug information into the debugger.

7. In the Confirm Perspective Switch dialog that appears, click Yes. DS-5 connects to the Altera target and displays the connection status in the Debug Control view.

**Figure 7-6: Hello World GCC debug configuration connected**

Debugging on bare-metal targets using DS-5 and GCC
compiler

Document ID: 102609_0100_02_en
Version 1.0
Creating a DS-5 debug configuration and connecting to your Altera
target

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

8. Click ▶ in the `Debug Control` view to continue running the application. You can view the application output in the `App Console` view.

**Figure 7-7: Hello World application console output**