Bare Metal User Guide

2015.11.30

UG-01165

Subscribe Send Feedback

Introduction

This guide will provide examples of how to create and debug Bare Metal projects using the ARM® DS-5 Altera Edition included in the *Altera® SoC Embedded Design Suite (SoC EDS) User Guide*.

The Altera SoC EDS is a comprehensive tool suite for embedded software development on Altera SoC devices. It includes the hardware abstraction library (HWLibs), ARM DS-5 Altera Edition (DS-5 AE), tool chain, and examples for a Bare Metal development environment.

The DS-5 AE is a useful toolset that allows you to create Bare Metal applications within the DS-5 IDE, configure, and execute it on the Altera SoC target board.

For more information, refer to the "Introduction to the SoC Embedded Design Suite" and "ARM DS-5 Altera Edition" sections in the *Altera SoC Embedded Design Suite User Guide*.

Refer to the additional guidelines, such as respective *SoC Development Kit User Guide* and *USB-Blaster User Guide* when you go through the examples in this guide.

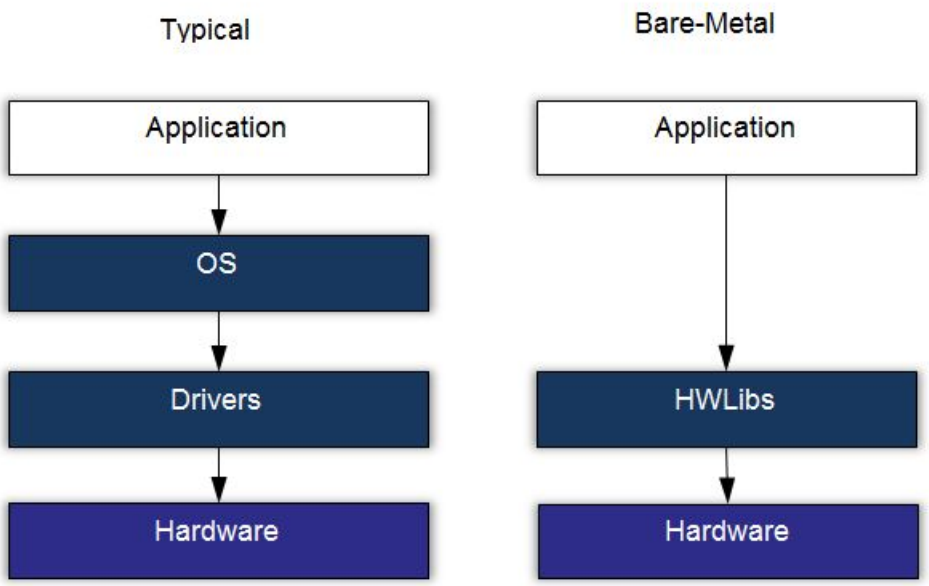
Related Information

- **Cyclone V SoC Development Kit User Guide**
- **Arria V SoC Development Kit User Guide**
- **Arria 10 SoC Development Kit User Guide**
- **USB Blaster Download Cable User Guide**
- **Altera SoC Embedded Design Suite User Guide**

Bare Metal Overview

Firmware applications intended to run without an operating system (OS) are referred to as Bare Metal applications. In comparison with the user application, which is managed by an OS, a Bare Metal application can interface directly to the system hardware and run without an OS.

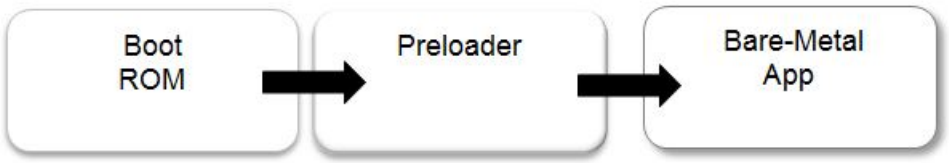
Figure 1: Bare Metal Application



The Bare Metal application can be invoked in one of many ways. In the following three scenarios, it is invoked after the Preloader boot stage has completed the system hardware initialization and verified the Bare Metal image or has been built as a Boot Module.

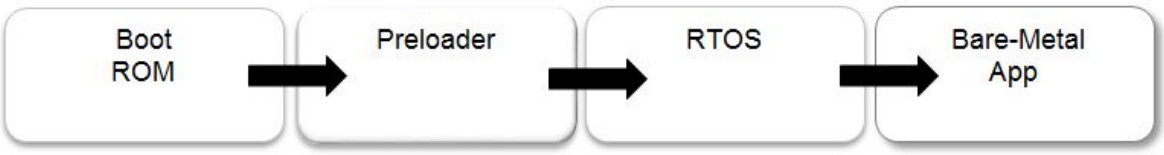
- Typical Bare Metal Application - when the Bare Metal application runs directly from the Preloader

Figure 2: Typical Bare Metal Application



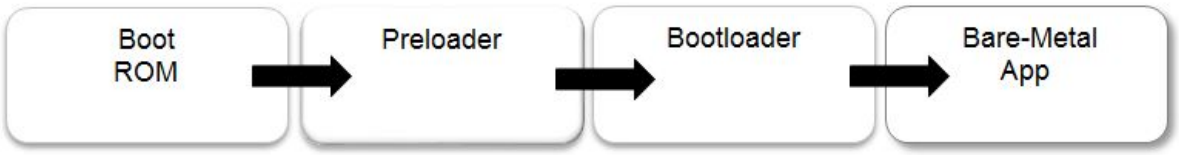
- RTOS Bare Metal Application - when the Bare Metal application runs from an RTOS

Figure 3: RTOS Bare Metal Application



- Bootloader Bare Metal Application - when the Bare Metal application runs from the Bootlooder

Figure 4: Bootloader Bare Metal Application



The *Altera SoC Embedded Design Suite (SoC EDS) User Guide* provides HW abstraction Application Programming Interfaces (APIs) to simplify Bare Metal application development.

Related Information

- **AN 709 HPS SoC Boot Guide**
  For more information about boot stages.
- **Minimal Preloader Example Project** on page 57
  For more information about the Bare Metal application built as a Boot Module.
- **Altera SoC Embedded Design Suite User Guide**

Prerequisites for the Bare Metal Development Environment

The following tools need to be installed before proceeding:

- Altera SoC EDS
- USB-Blaster driver

Note: The USB-Blaster must be connected to the board and Altera SoC EDS license file must be setup correctly before proceeding.

The Altera SoC EDS provides the following components for a complete Bare Metal Software Development Environment:

- ARM DS-5 Altera Edition
  - ARM Compiler 5
  - GNU Compiler Collection (GCC) Bare Metal Compiler
- HWLibs
- Mkpimage tool (required by BootROM)
- Mkimage tool (required by Preloader)
- SD card image tool
- Golden Hardware Reference Design (GHRD)

For more information, refer to the "Installing the Altera SoC Embedded Design Suite" and "ARM DS-5 Altera Edition" sections in the *Altera SoC Embedded Design Suite User Guide*.

Related Information

Altera SoC Embedded Design Suite User Guide

Bare Metal Compiler

The Bare Metal Compiler that is shipped with the Altera SoC EDS is the Mentor Graphics® Sourcery™ Code Bench Lite Edition. The compiler is a GCC-based **arm-altera-eabi** port. It targets the ARM processor, it assumes bare metal operation, and it uses the standard ARM embedded application binary interface (EABI) conventions. The bare metal compiler is installed as part of the Altera SoC EDS installation.

There are 2 types of bare metal compilers provided:

- ARM Compiler
- GNU Compiler Collection (GCC)

ARM compiler is supported by the Full ARM DS-5 edition (for all ARM processors) that requires a license, while the GCC is provided by DS-5 Altera Editions which is free.

For more information on the Bare Metal Compiler, refer to the "Bare Metal Compiler" chapter in the *Altera SoC Embedded Design User Guide*.

Related Information

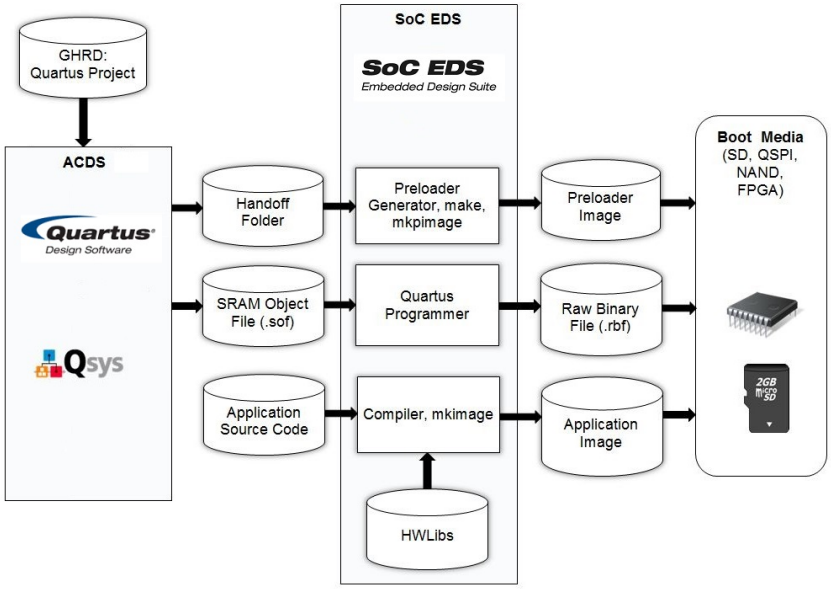
Altera SoC Embedded Design Suite User Guide

Bare Metal Development Flow

Developing SoC based Bare Metal applications involve dependencies from the FPGA generated design tools and use of the Altera SoC EDS packaged tools for building and debugging the application.

A typical Bare Metal Development flow is shown below:

Figure 5: Typical Bare Metal Flow



Using DS-5 AE to Create and Manage Bare Metal Projects

Bare Metal application project can be created using DS-5 AE and compiled with ARM or GCC compiler. The application can be created to run from various target such as On-Chip RAM (OCRAM) or external memory.

The following sections will guide you through how to create, build, load and debug a simple bare metal project named **"Hello World"** on Cyclone V SoC Development Kit using the ARM compiler:

- To run from OCRAM
- To run from SDRAM

Note: For the GCC compiler, you can import an existing bare metal project example compiled using GCC compiler or refer to "GCC Bare-Metal Project Management" to create a simple C project manually.

Related Information

- **GCC Bare-Metal Project Management**
- **Getting Started with Bare Metal Project Management**

Simple Bare Metal Project Using On-Chip-RAM

In the following sections, you are creating, building, loading, and debugging a simple **"Hello World"** application project to run from OCRAM using ARM compiler.

Related Information

Altera SoC Embedded Design Suite User Guide

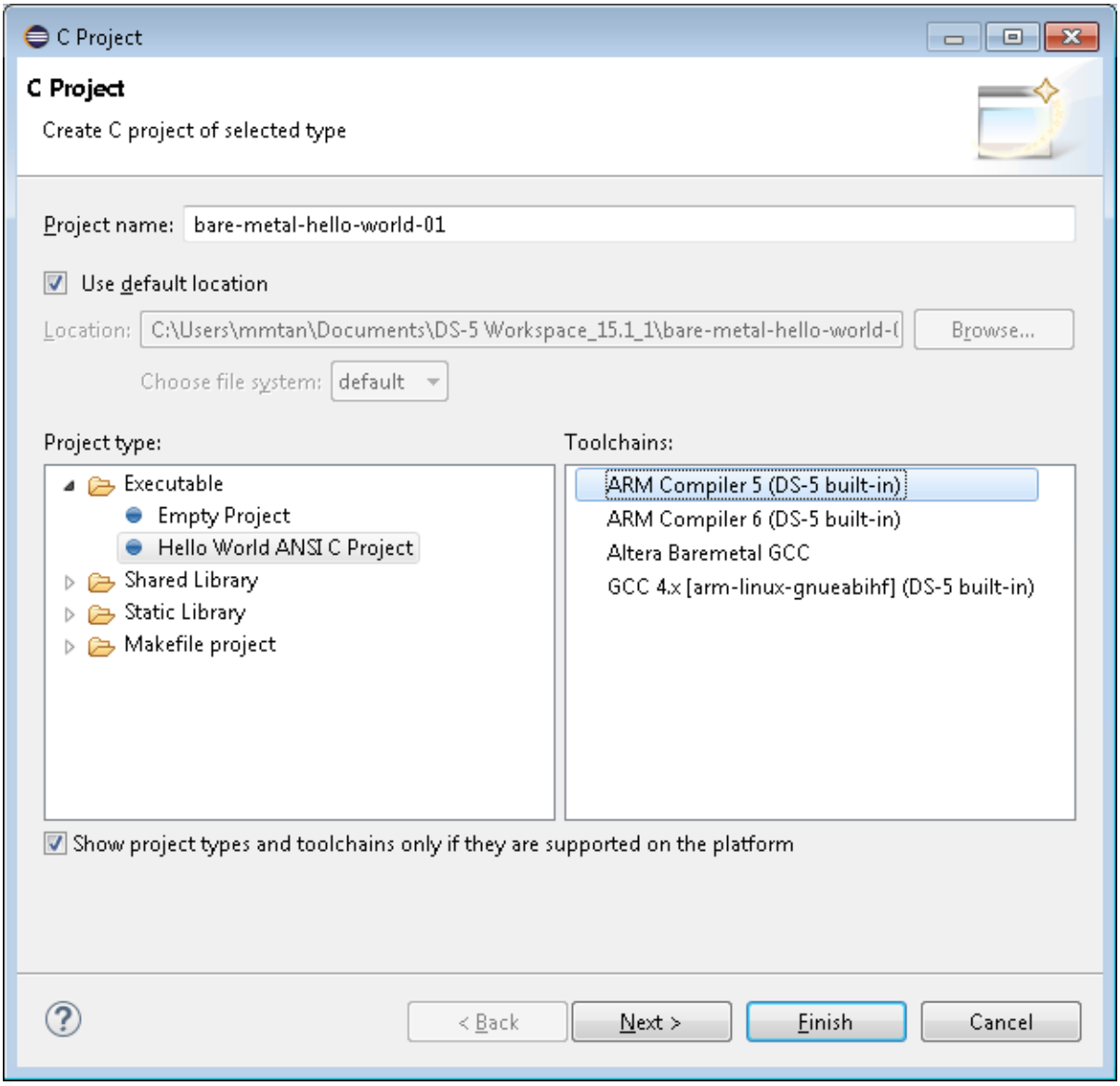
Create Project

Before you begin

In Windows, go to **Windows** > **All Programs** > **ARM DS-5** > **Eclipse for DS-5** to open the ARM DS-5 tool. Select a workspace before you begin. If it is not already selected, change to **C/C++ Perspective**, located at the top right tabs of DS-5.

1. Create a new C project by selecting **File** > **New** > **C Project**.
2. Select **Project Type** as "Hello World ANSI C Project" and **Toolchains** as "ARM Compiler 5 (DS-5 built-in)" and enter a unique project name in the **Project Name** field. For example, bare-metal-hello-world-01.

Figure 6: Creating C Project of Selected Type



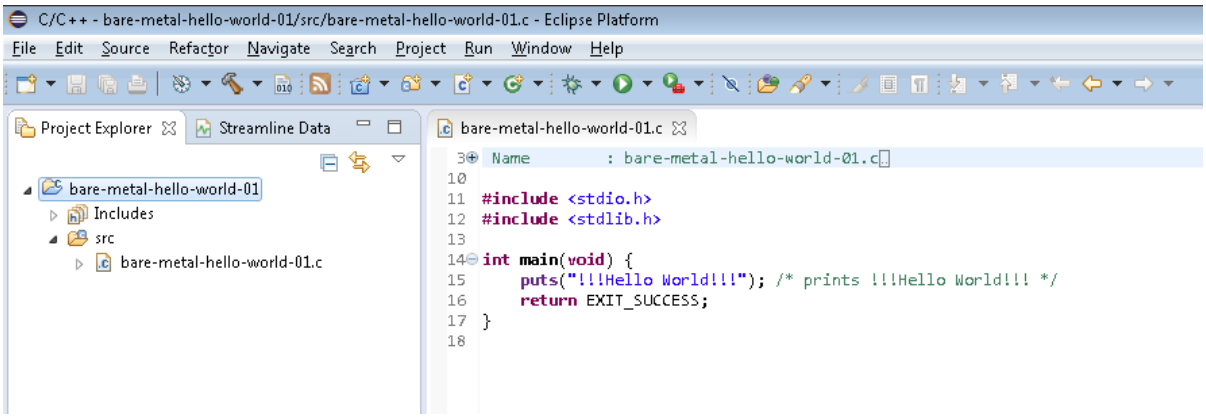
Note: The DS-5 is supplied with two versions of the ARM Compiler for compiling bare metal applications. ARM Compiler 5 supports all ARM architectures except ARMv8. ARM Compiler 6 supports architectures ARMv8 and ARMv7-A, as well as alpha support for architectures ARMv7-R, ARMv7-M and ARMv6-M. For Altera SoC FPGA, ARM Compiler 5 is required.

Both versions of ARM Compiler are license managed and not all editions of DS-5 include a license for it.

For any licensing information, please refer to the "Licensing" chapter in the *Altera SoC EDS User Guide*.

3. Select **Finish**.
   The source code for `bare-metal-hello-world-01.c` appears in the editor view.

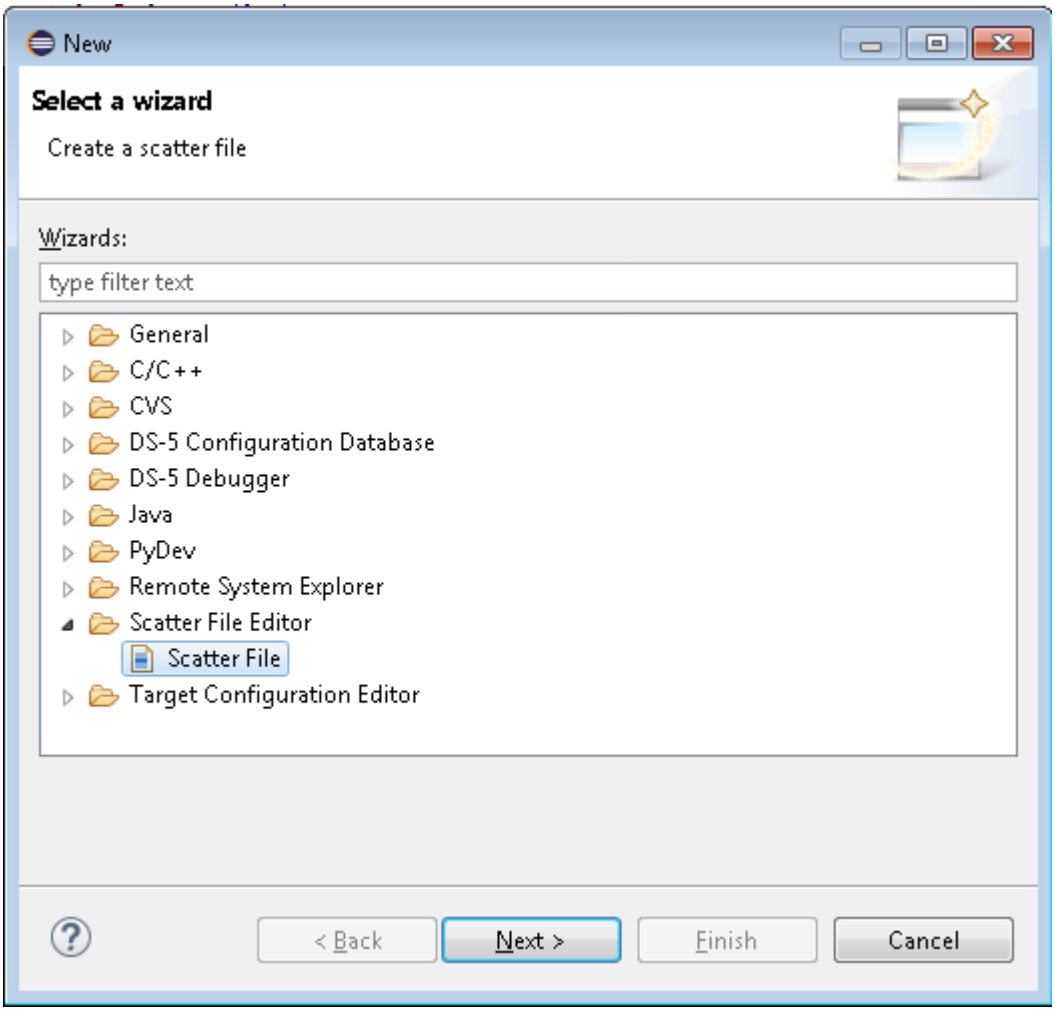
Figure 7: Bare Metal "Hello World - 01" Code Snippet



Create New Scatter File to Locate the Bare Metal Application in the OCRAM

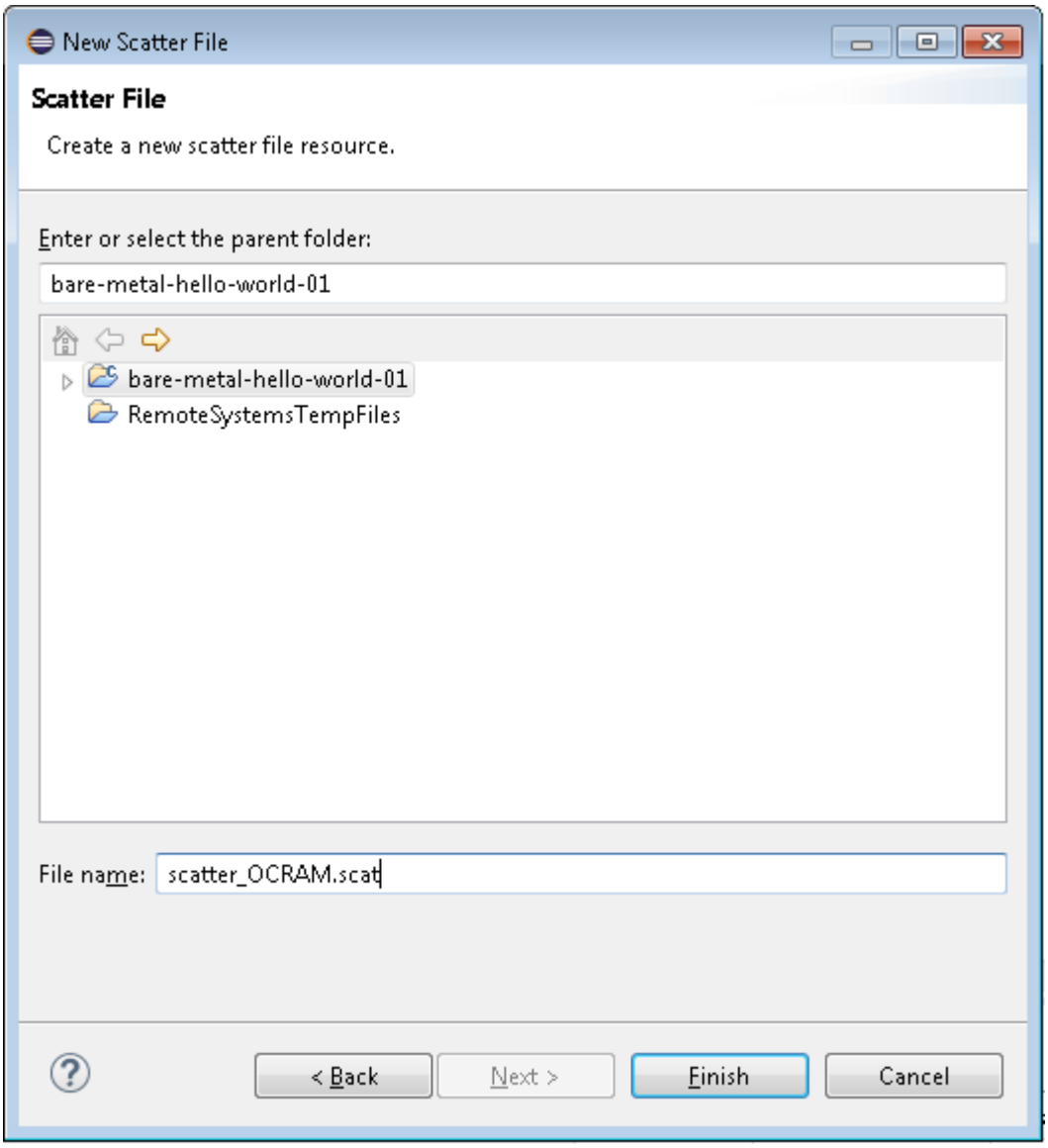
1. Create a scatter file. Right click on the project, and select "**New** > **Other...**", then "**Scatter File Editor** > **Scatter File**".
   The scatter file enables you to specify the memory map of an image to the linker using a description in a text file. It is used by the ARM compiler linker to determine the placement of the program in the target memory.

Figure 8: Creating Scatter File



2. Select the project name, `bare-metal-hello-world-01`, and enter the scatter file name, like `scatter_OCRAM.scat`

Figure 9: Scatter File Setting



3. Select **Finish**.
   The new file automatically appears in the `Project Explorer` view.
4. In the `scatter_OCRAM.scat` editor view, enter the following when targeting a Cyclone V or Arria V device:

```
OCRAM 0xFFFF0000 0x10000
{
APP_CODE +0
{
* (+RO, +RW, +ZI)
}
ARM_LIB_STACKHEAP 0xFFFF8000 EMPTY 0x8000 ; Application help and stack
{ }
}
```

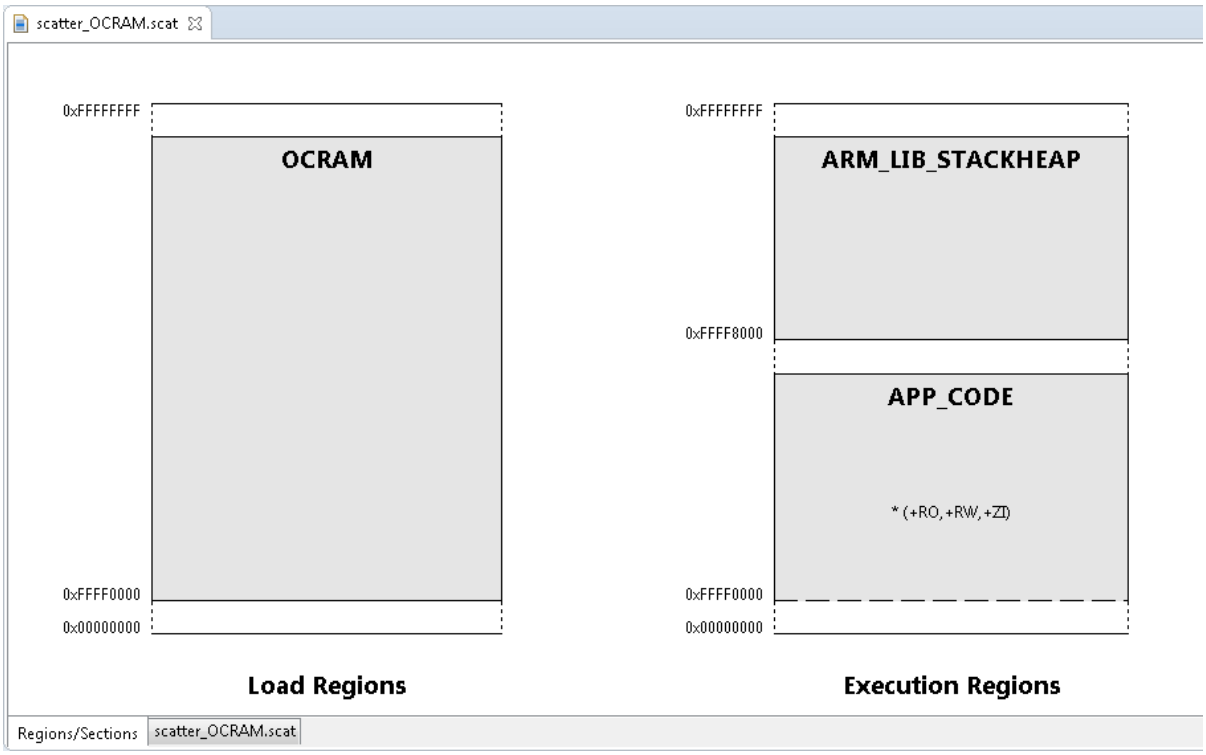
The linker script instructs the linker on how to link the application:

- Defines OCRAM base address (`0xFFFF0000`) and size (`0x10000`)
- Loads all application sections in the OCRAM
- Allocates a maximum of 32 KB (`0x8000`) for stack and heap starting from address `0xFFFF8000`

The parameters can easily be changed for targeting Arria 10 devices, where there are 256 KB of OCRAM located at `0xFFE00000`.

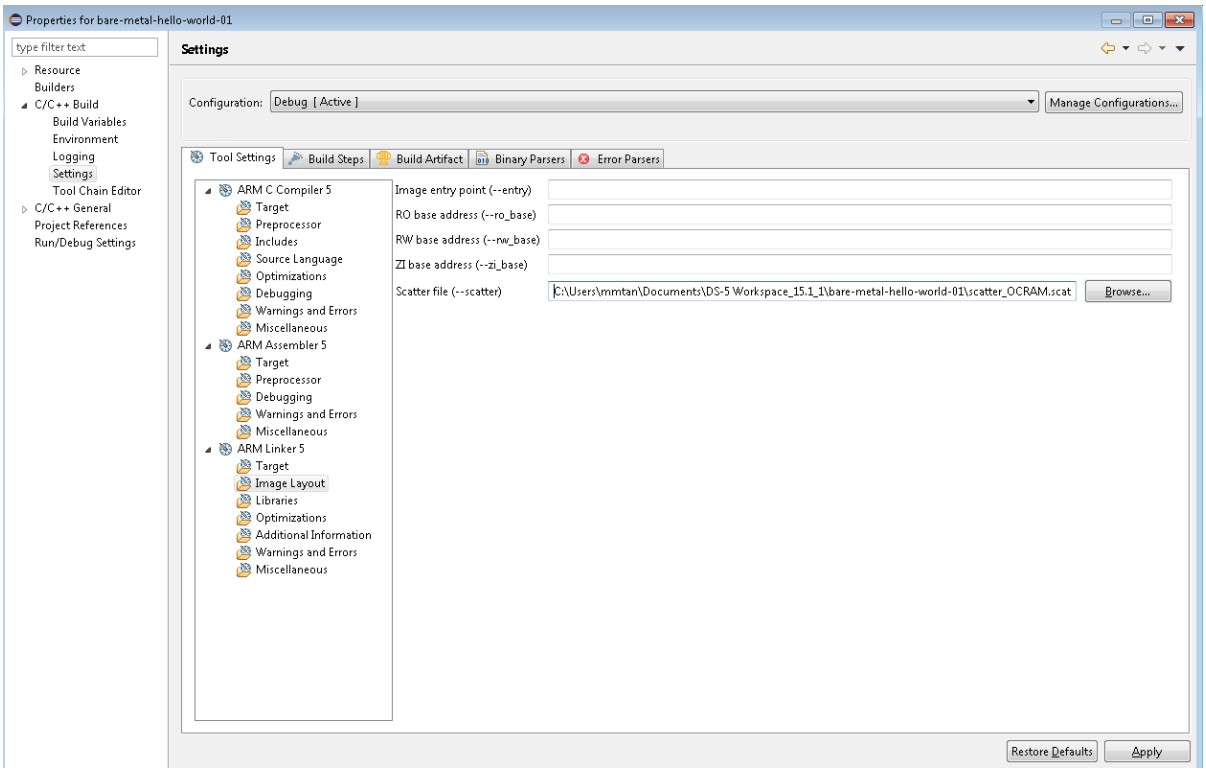
5. Select the **Regions/Sections** tab, located just below the scatter file to show what the memory map looks like.

Figure 10: Scatter File Regions



6. Select **File** > **Save** to save the modifications.
7. After the scatter file has been created in the project, it needs to be associated with the project properties. Select the project name in the `Project Explorer` view and right-click to select **Properties**.
8. Go to **C/C++ Build** > **Settings** > **Tool Settings** > **ARM Linker 5** > **Image Layout**.
9. In the **Scatter file (--scatter)** text field, browse to the newly created scatter file which should now be in the project folder.
10. Select **Apply** and then **OK**.

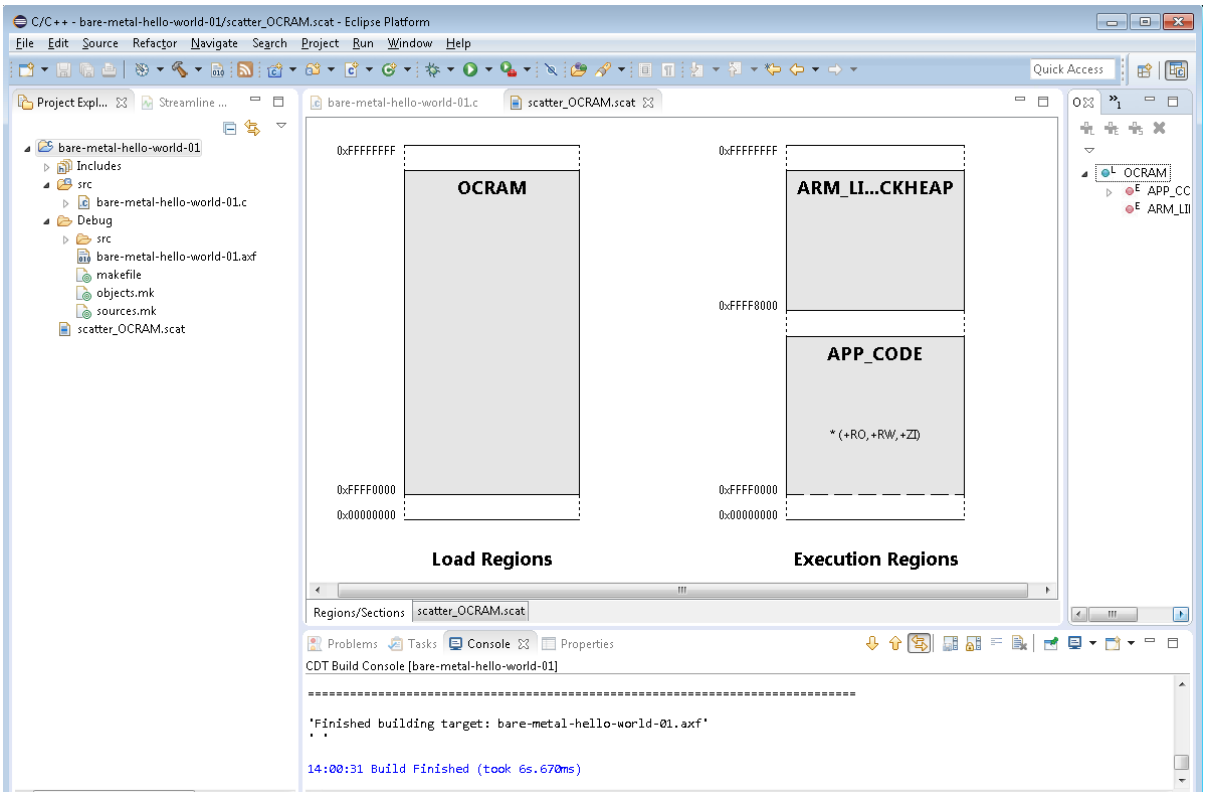
Figure 11: Scatter File Location Setting



Build and Debug the Project

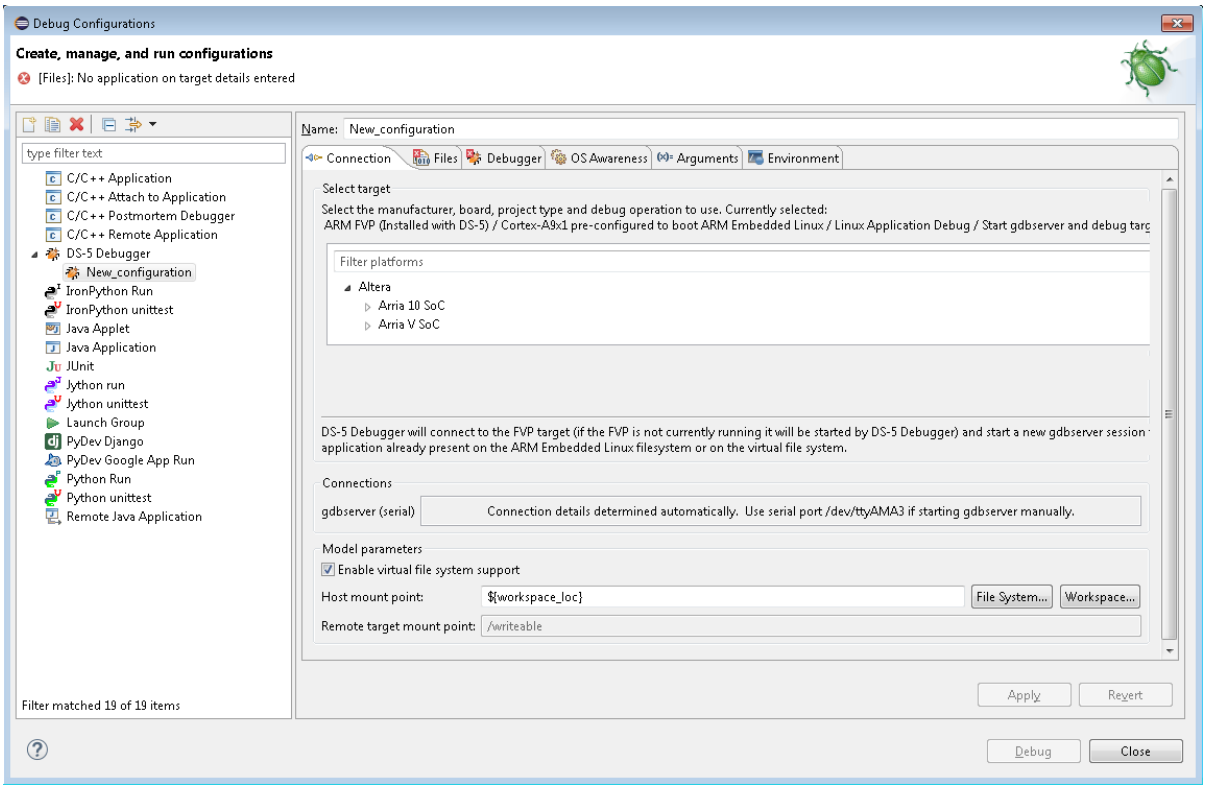
1. Then, right click on the project and select "Build Project".
   The “Debug” build directory will be created and the compiled object modules will be placed there.

Figure 12: Bare Metal "Hello World - 01" Built Done



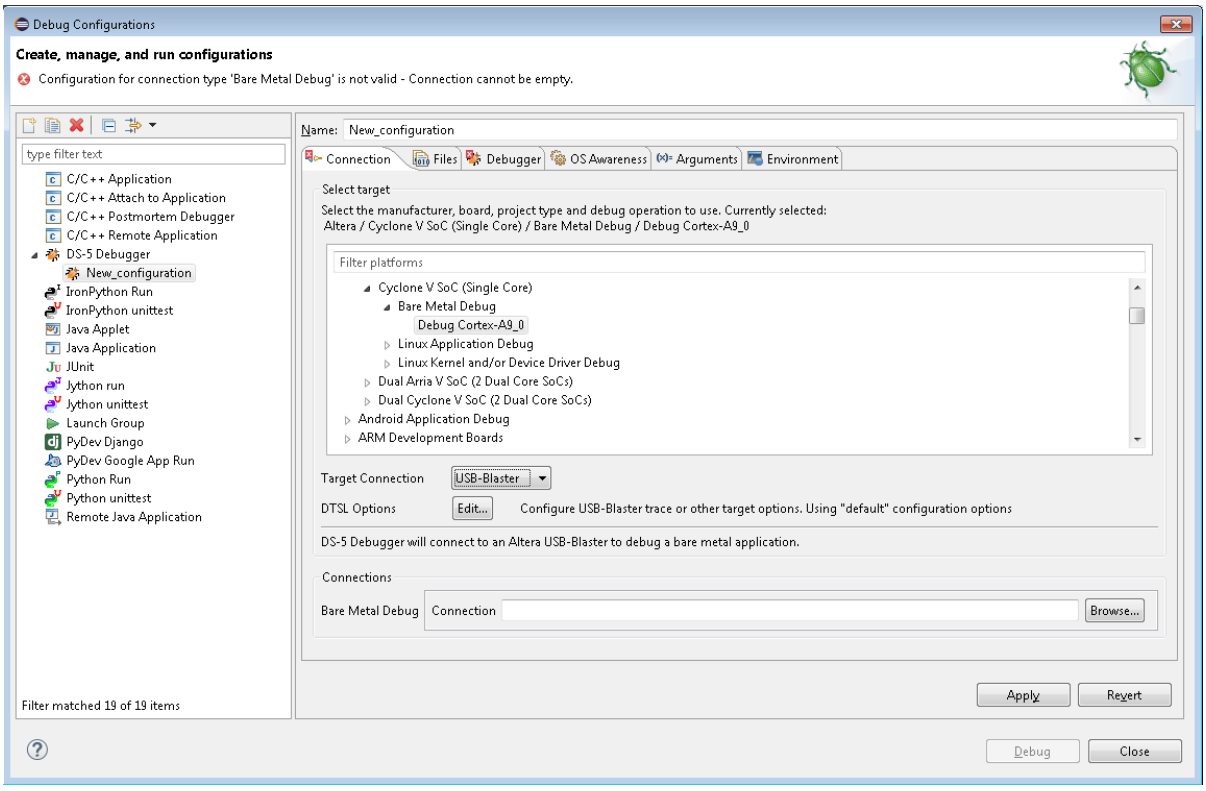
2. To download and debug the program, select **Run** > **Debug Configurations**.
3. Right click on DS-5 Debugger and select **New** to create a new debug configuration.

Figure 13: New Debug Configuration



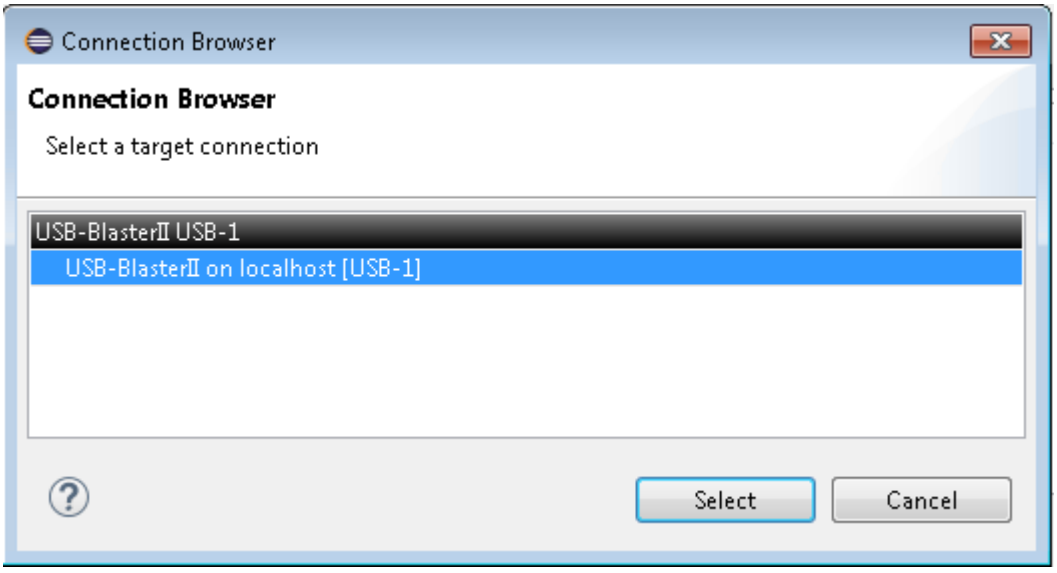
4. In the **Connection** tab, select **Altera** > **Cyclone V SoC (Single Core)** > **Bare Metal Debug** > **Debug Cortex-A9_0** and select "USB-Blaster" from the **Target Connection** pull-down menu.

Figure 14: New Debug Configuration Target Connection Setting



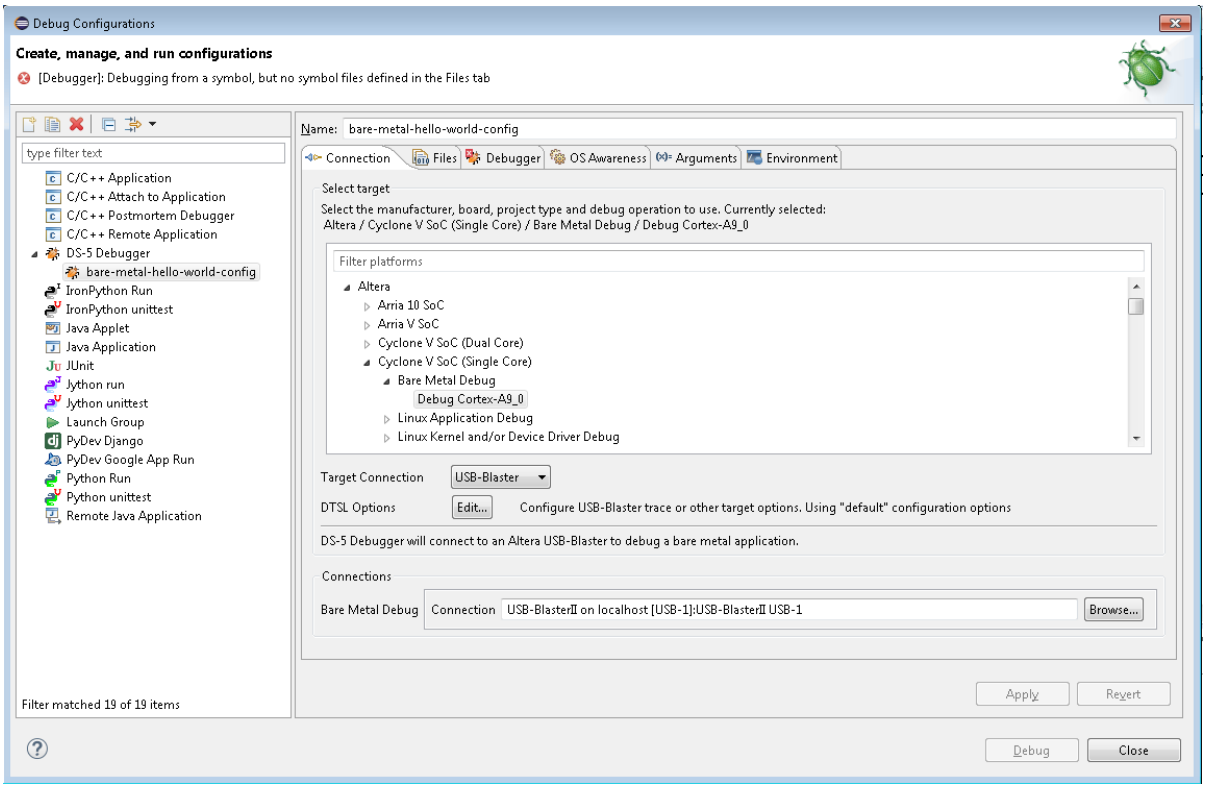
5. Select an available Bare Metal Debug Connection by clicking **Browse**.
   This will return a list of the available debug connections.

Figure 15: Connection Browser



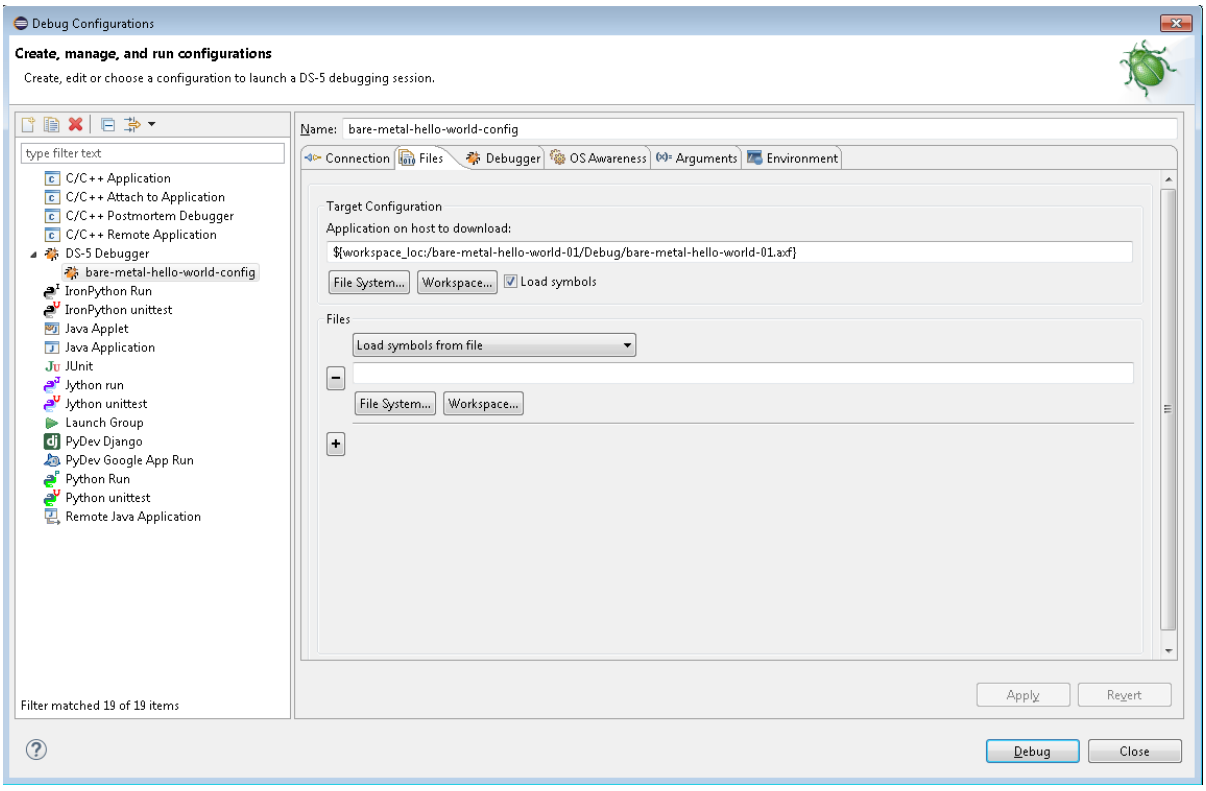
6. Select the hardware and then click **Select**. Change the name of the configuration from "New_Configuration" to "bare-metal-hello-world-config", and select **Apply**.

Figure 16: Bare Metal Hello World Config Connection Settings



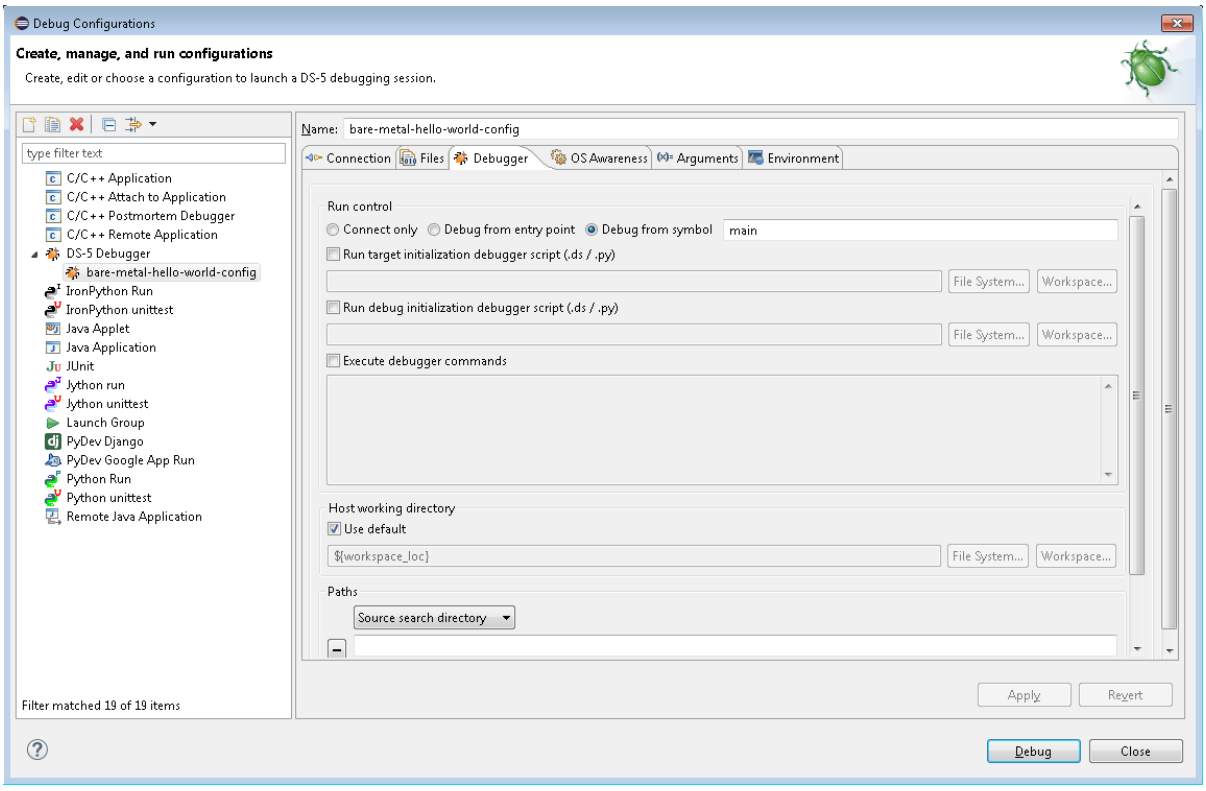
7. Go to the **Files** tab and select the application to download by clicking “Workspace” and select the object module that was built, `bare-metal-hello-world-01.axf`.
   It should be in the Debug folder within the project. Make sure that “Load symbols” is selected, and then click on **Apply**.

Figure 17: Bare Metal Hello World Config Files Settings



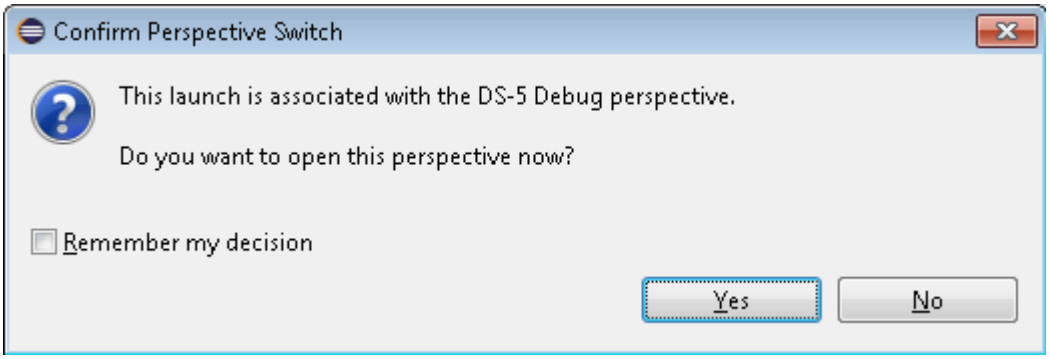
8. Go to the **Debugger** tab and make sure that under **Run control**, **Debug from symbol** is set to "main".

Figure 18: Bare Metal Hello World Config Debugger Settings



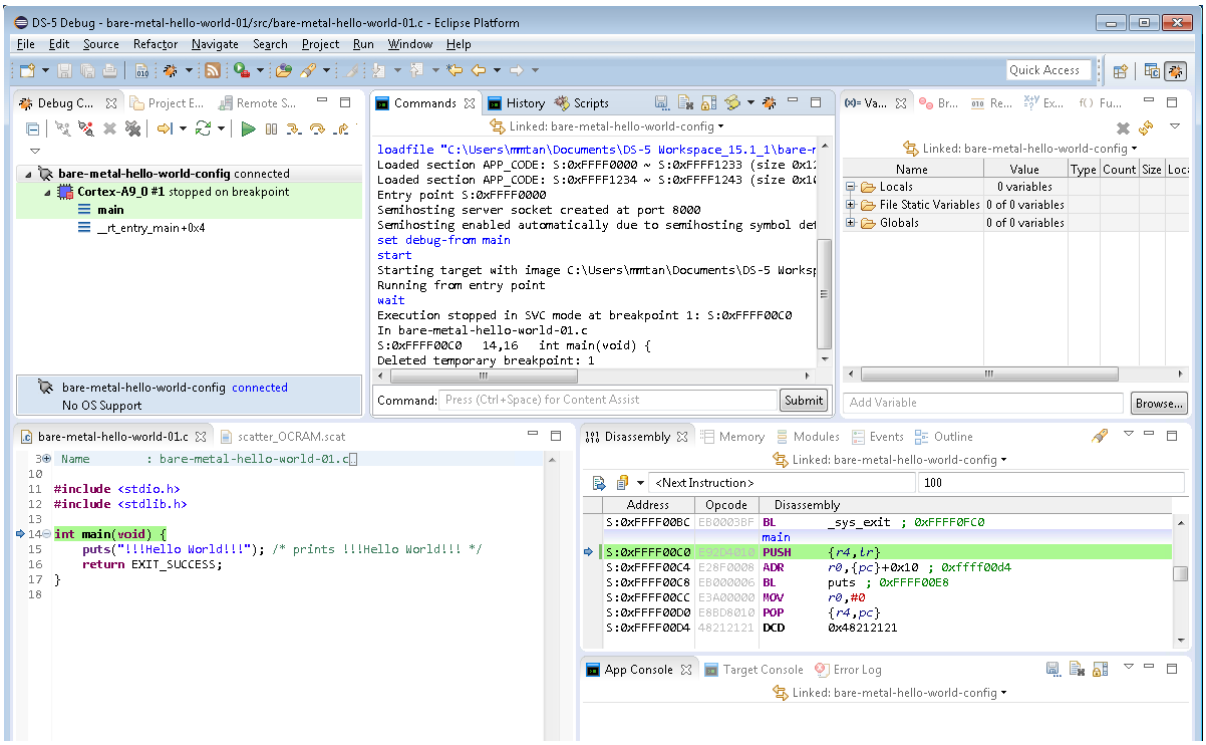
9. Select **Debug** to load and debug the application.
   It runs a query to change to the `DS-5 Debug Perspective`. Select **Yes**.

Figure 19: Confirm Perspective Switch



10. It switches to the `DS-5 Debug Perspective` and then download and begin to run the application.
    The program stops at main and waits there. In the `Commands` view, notice the entry point is **0xFFFF0000**, which is the start address specified in the scatter file for the on-chip RAM.

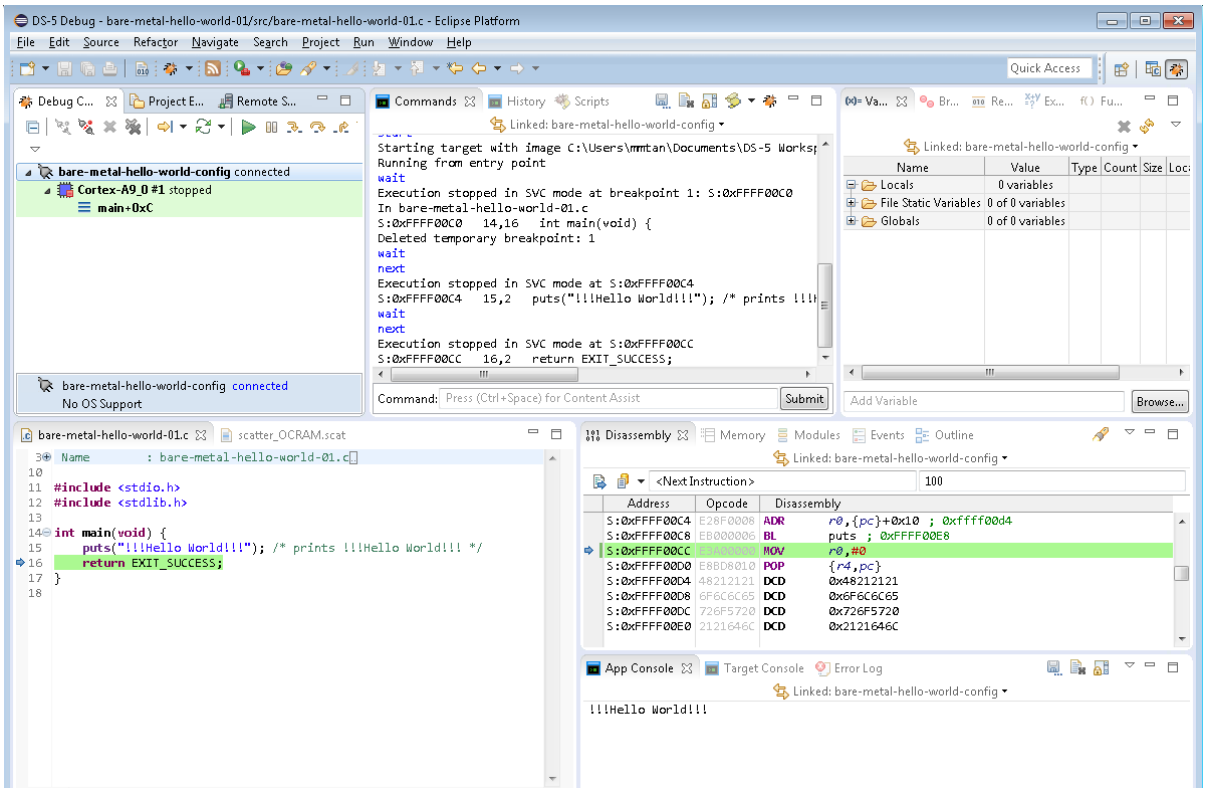
Figure 20: OCRAM Debug Stop at Main



11. Click the **Step Over Source Line** icon or press **F6**.
12. Then "Step" again.

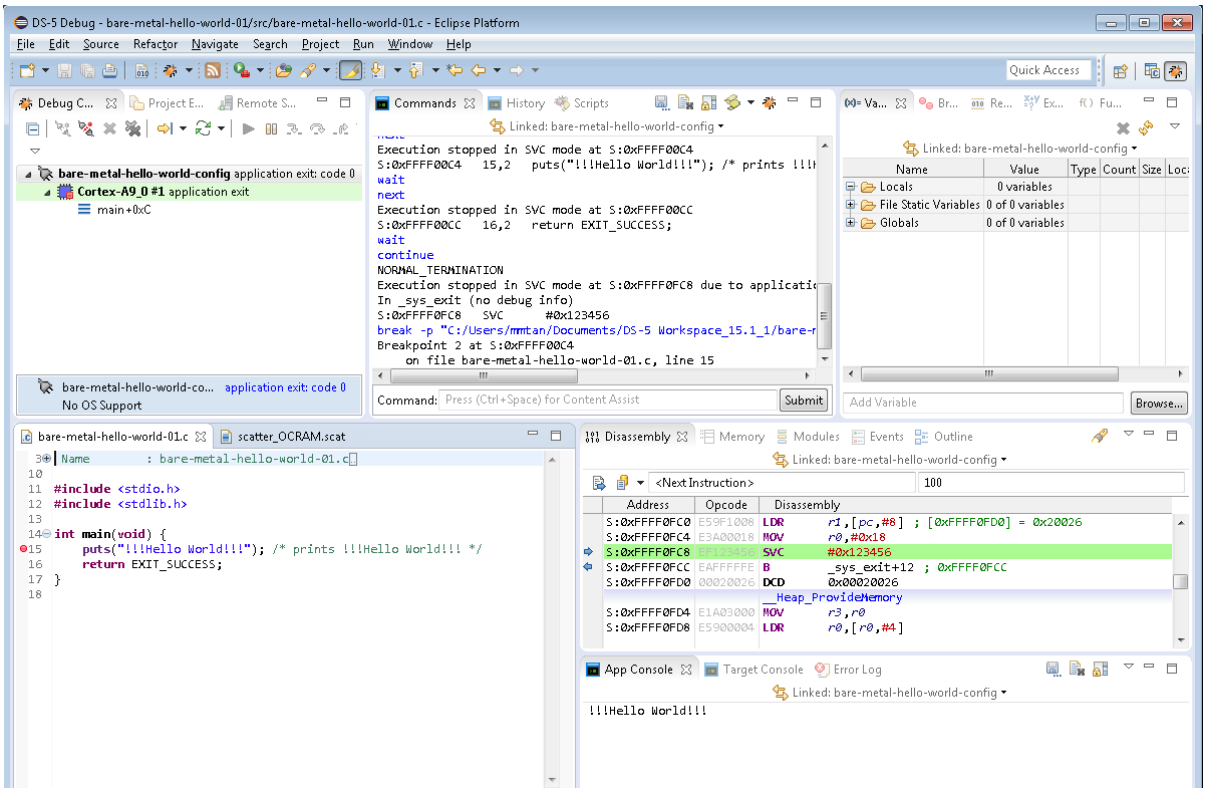
    The output appears on the `App Console` view.

Figure 21: "App Console" Output



13. Select **Continue** to exit out of the application.
14. Create a breakpoint by double clicking on the left margin of the `bare-metal-hello-world-0.1.c` `Source` view. If the `Breakpoints` view is not shown, then open it by selecting: **View** > **Show View** > **Breakpoints**.

Figure 22: Debug Breakpoint Setup



Preloader

The Preloader is a boot-strap program that configures some components of the board including the memory controller during start up. It normally runs automatically if it were burned into one of the Flash devices on the board.

For testing purposes or before the Preloader has been burned into Flash, the Preloader can be run by downloading and executing it from the DS-5.

Loading and Running the Default Preloader

There are a few options for running the Preloader.

- It can be ran through a DS-5 command script.
- It can be imported into a DS-5 project and launched similarly to any other Bare Metal application.
- It can be launched by the DS-5 **Run Control (Debug Control)** independent from the DS-5 projects.

This section demonstrates how the preloader is launched independently from the DS-5 project for Cyclone V SoC Development Kit. To optionally import the Preloader into a DS-5 project, refer to the “Importing Preloader into a DS-5 Project” section.

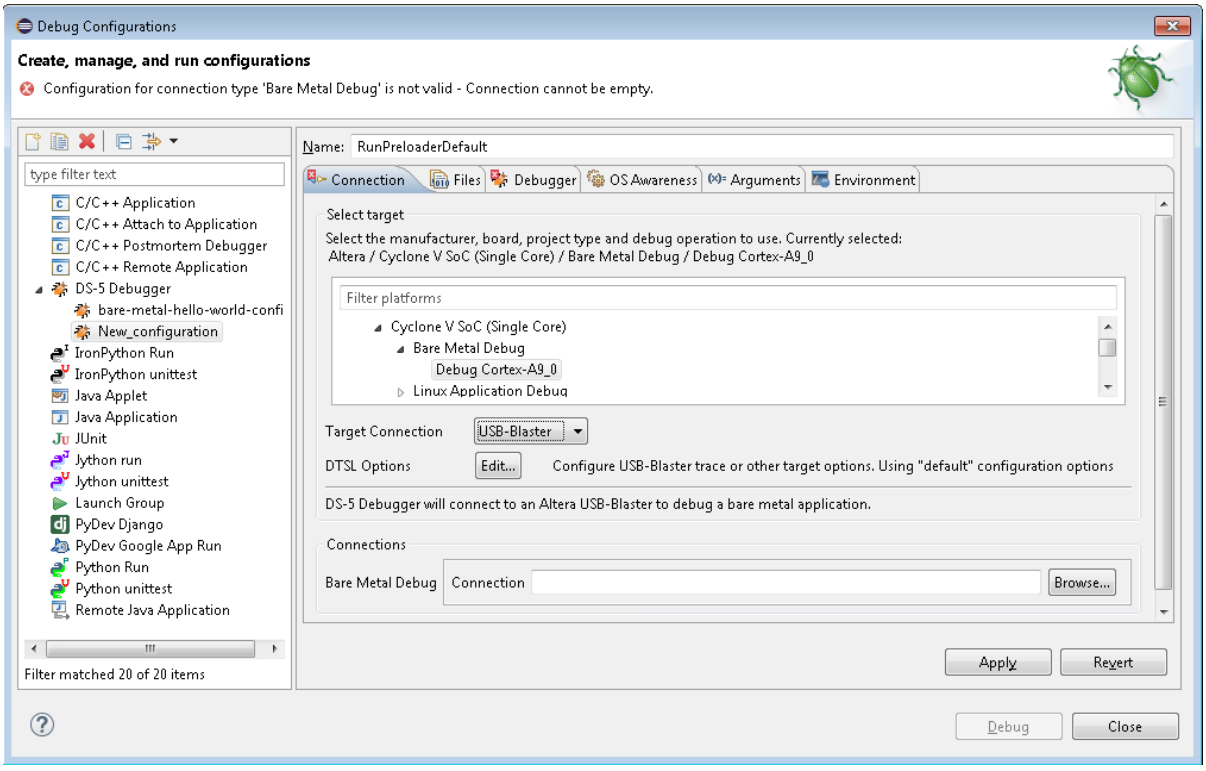
This process uses the **Run Configuration (Debug Configuration)** of DS-5 to download and execute the Preloader.

1. To begin, create a new **Debug Configuration**. **Run** > **Debug Configurations**.
2. Select **DS-5 Debugger** and click on **New** icon, which is located just above the "type filter text" field.

This creates a new **Debug Configuration** named "New_configuration". Change the name in the **Name** field to something like "RunPreloaderDefault".

3. On the **Connection** tab, select **Altera** > **Cyclone V SoC (Single Core)** > **Bare Metal Debug** > **Debug Cortex-A9** and then select "USB-Blaster" from the **Target Connection** pull-down menu.

Figure 23: RunPreloaderDefault Target Connection Setting



4. To select the physical debug connection, after the **Connections** field, click on **Browse** to select the specific Debug Hardware connection.

   Note: If there is only one debugger connected, then only one will show up in the list (as shown).

Figure 24: Connection Browser

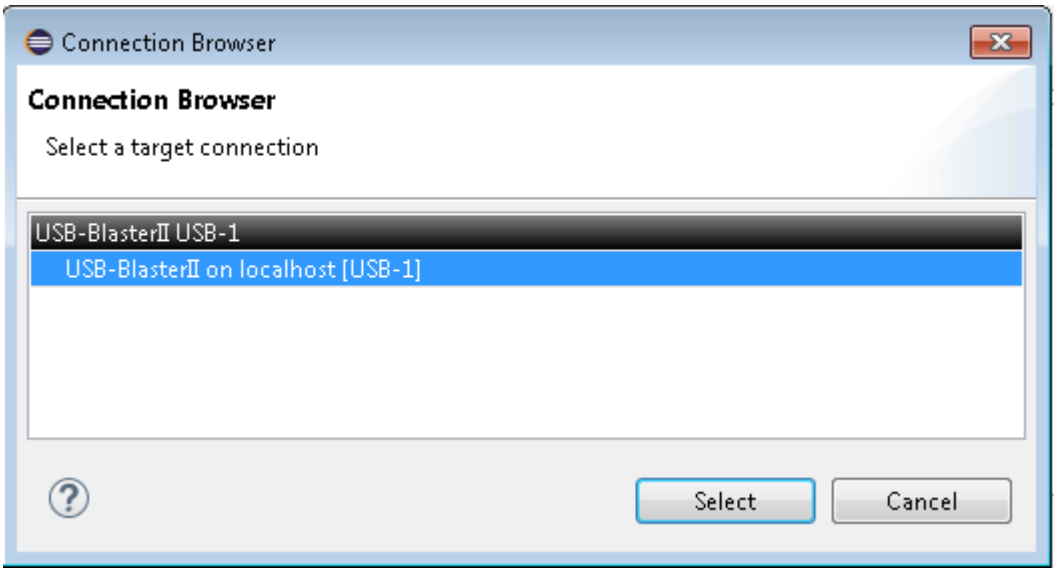
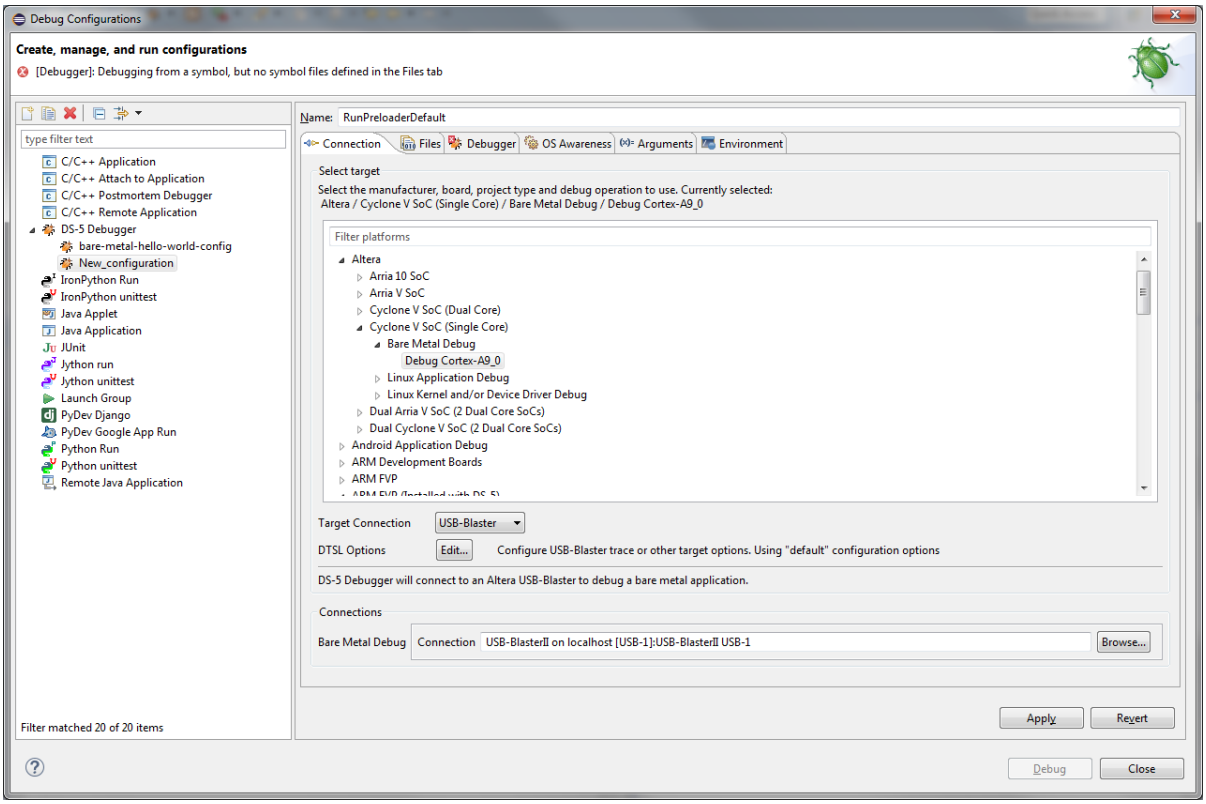


Figure 25: RunPreloaderDefault Connection Settings

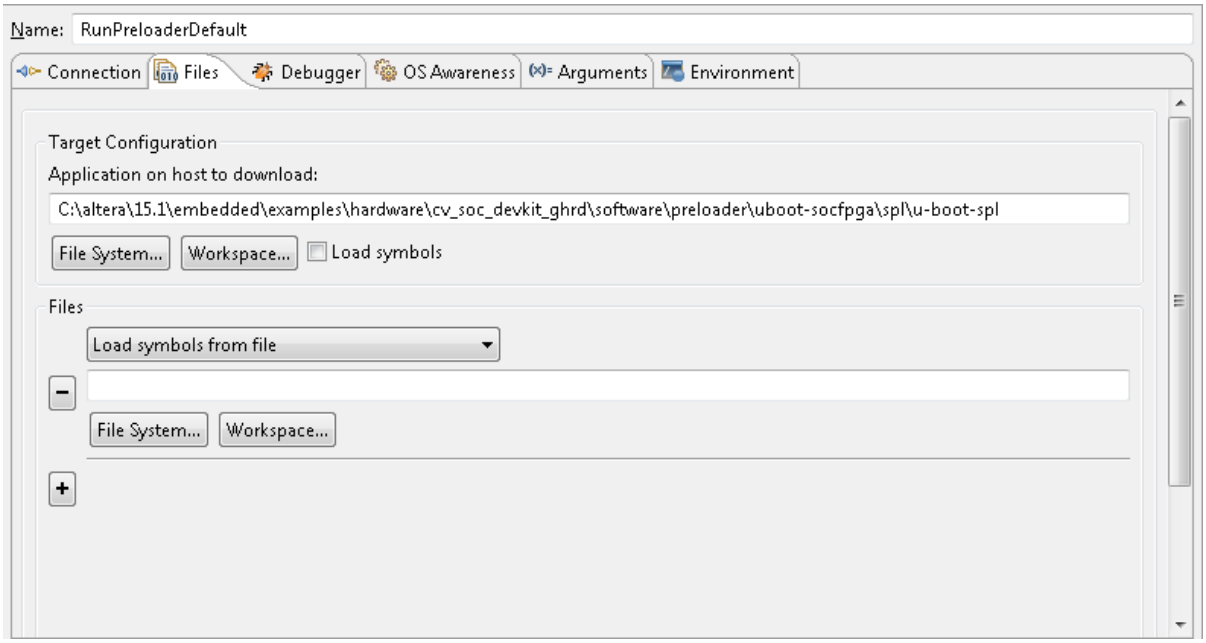


5. Select the **Files** tab and use "File System…" to browse to the Preloader image (**"u-boot-spl"**).

The Preloader image can typically be found in the following location: <*SoC EDS installation folder*>\`embedded\examples\hardware\cv_soc_devkit_ghrd\software\preloader\uboot-socfpga\spl\u-boot-spl`.

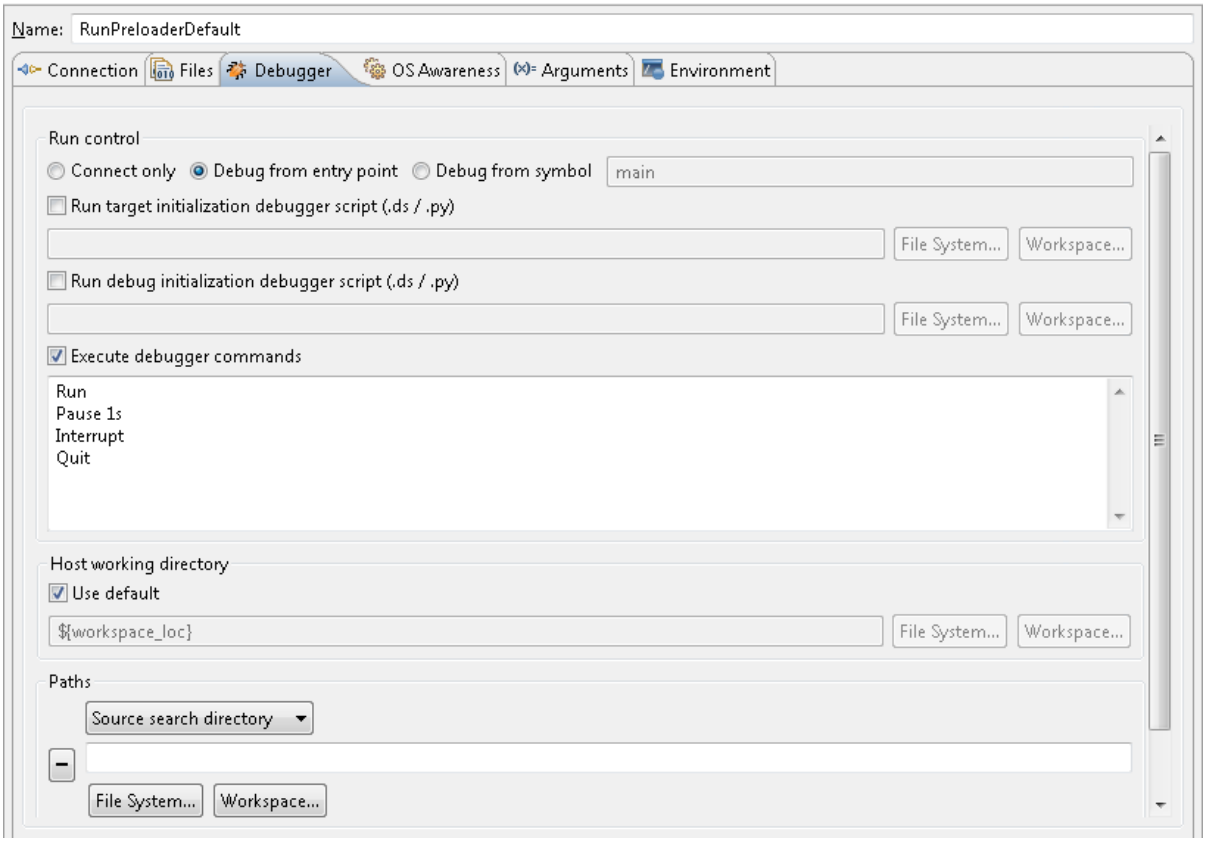
Note: Uncheck the "Load symbols" option.

Figure 26: RunPreloaderDefault Files Settings



6. On the **Debugger** tab, select "Debug from Entry Point" under **Run Control** and select **Execute debugger commands** to enable the following commands:
   - Run
   - Pause 1s
   - Interrupt
   - Quit

Figure 27: RunPreloaderDefault Debugger Settings



7. Select **Apply**, **Debug**, and then "Yes" to switch to the DS-5 Debug perspective, if queried.

Importing Preloader into a DS-5 Project

Instead of loading and running the default preloader directly, as described in the "Loading and Running the Default Preloader" chapter, you can import the Preloader executable **(u-boot-spl)** into your project within your workspace and browse to it by clicking on **Workspace**, as shown in the following sections. This isolates your project from any changes that you may make to the default preloader.

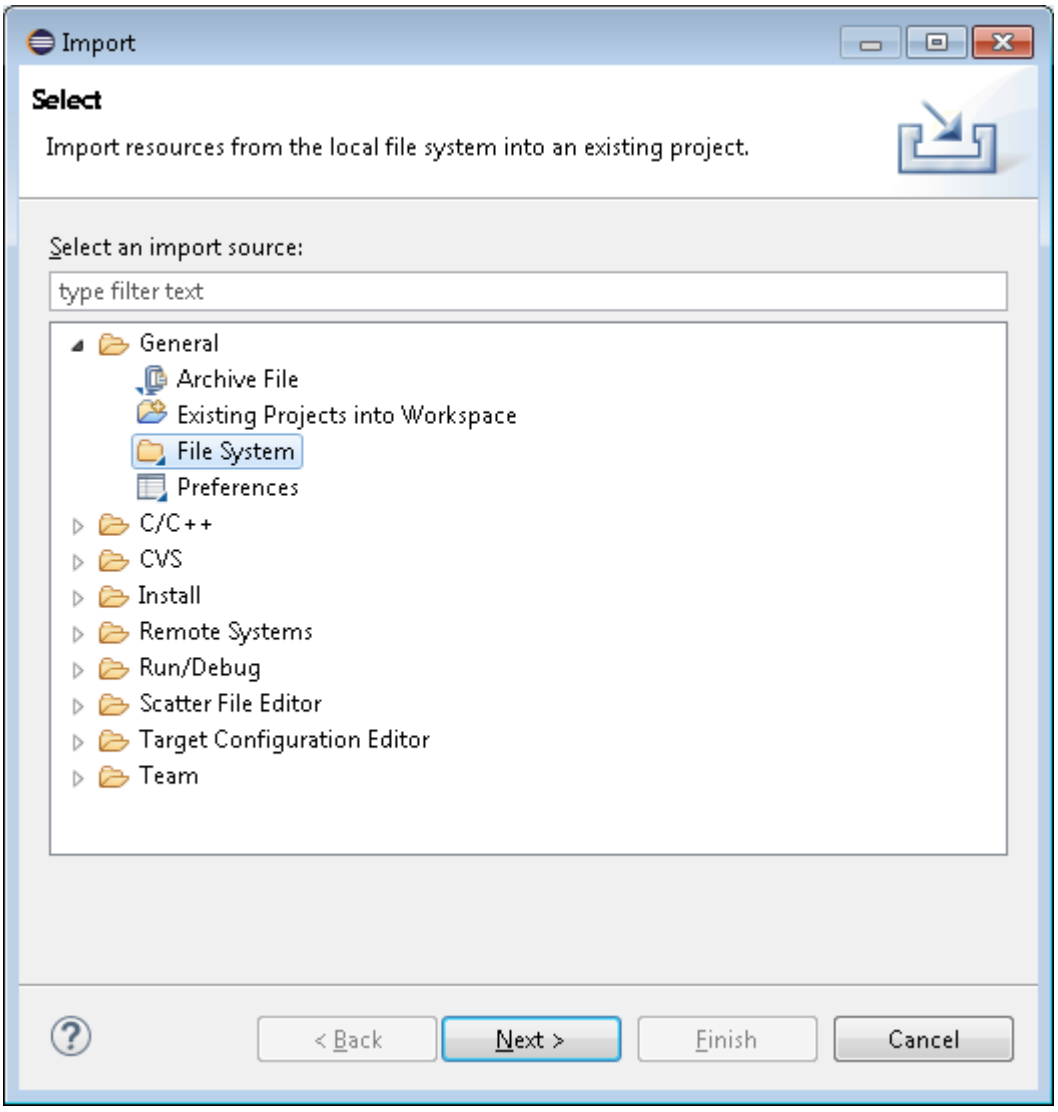
Import Preloader

Before you begin

To do so, first you have to import the Preloader into your project.

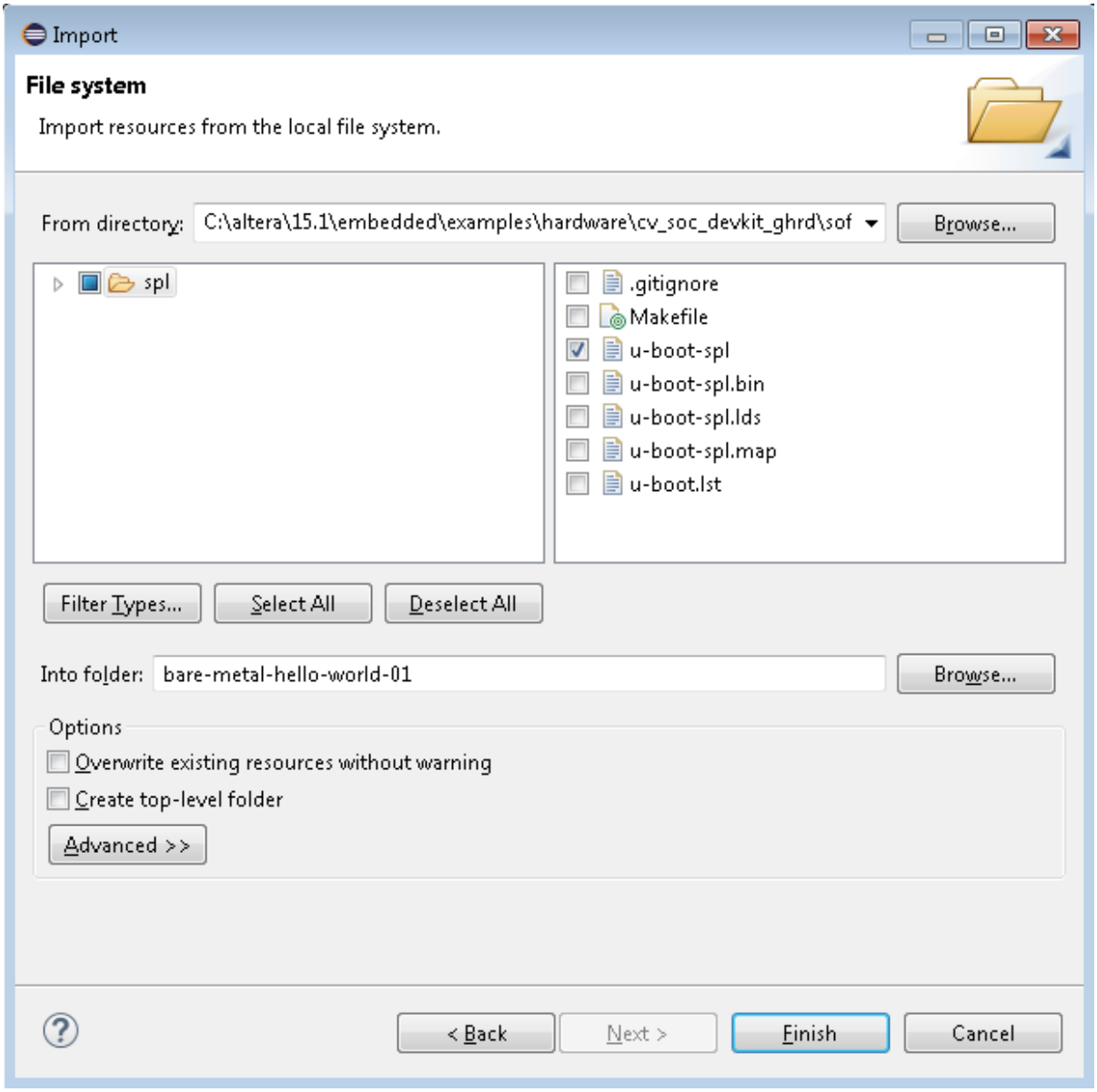
1. Right click on the project and select **Import**. Then select **General** > **File System** and click **Next** when you are done.

Figure 28: Import File System



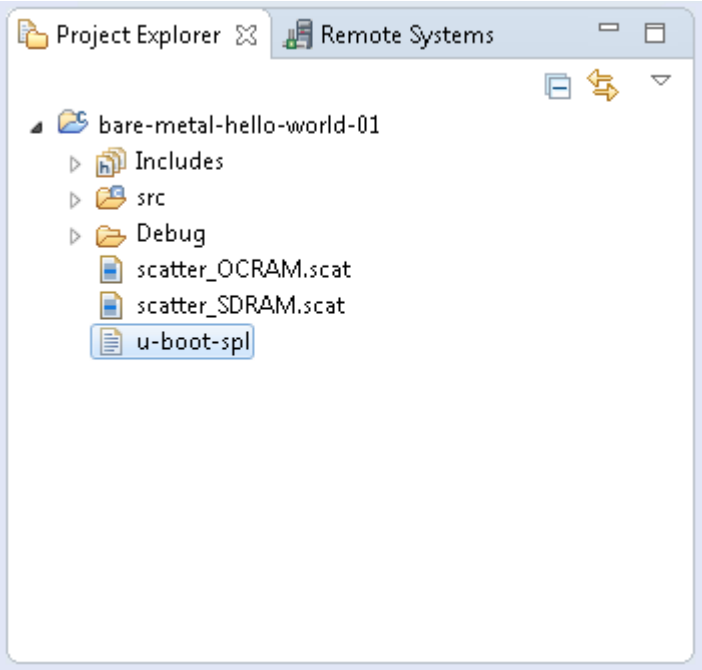
2. Browse to the Preloader executable **(u-boot-spl)** or enter the full path to the directory, for example: *<SoC EDS installation path>*`\embedded\examples\hardware\cv_soc_devkit_ghrd\software\preloader\uboot-socfpga\spl`.
3. Check the box next to the file name and then select **Finish**.

Figure 29: Import u-boot-spl



4. Make sure the newly imported file is shown in the Project Explorer.

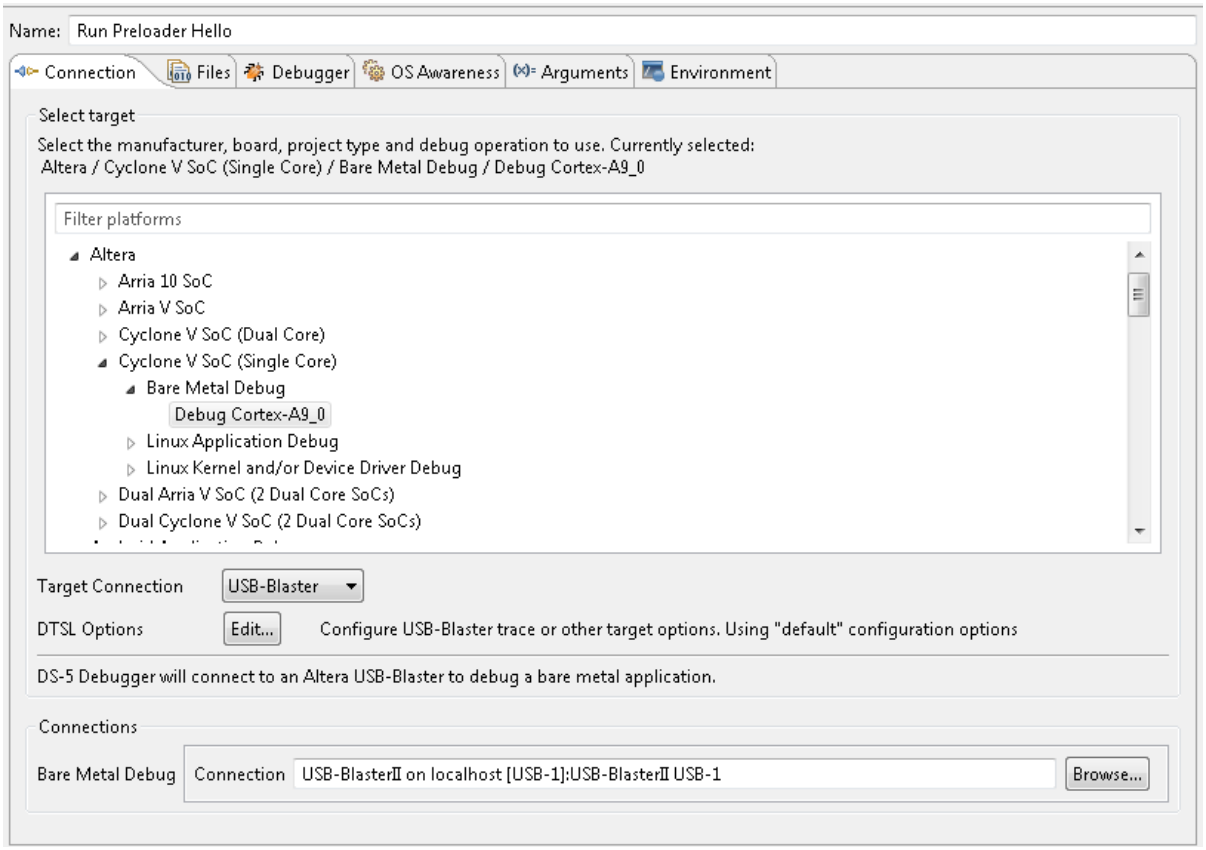
Figure 30: u-boot-spl file in Project Explorer



Create New Debug Configurations and Debug Preloader

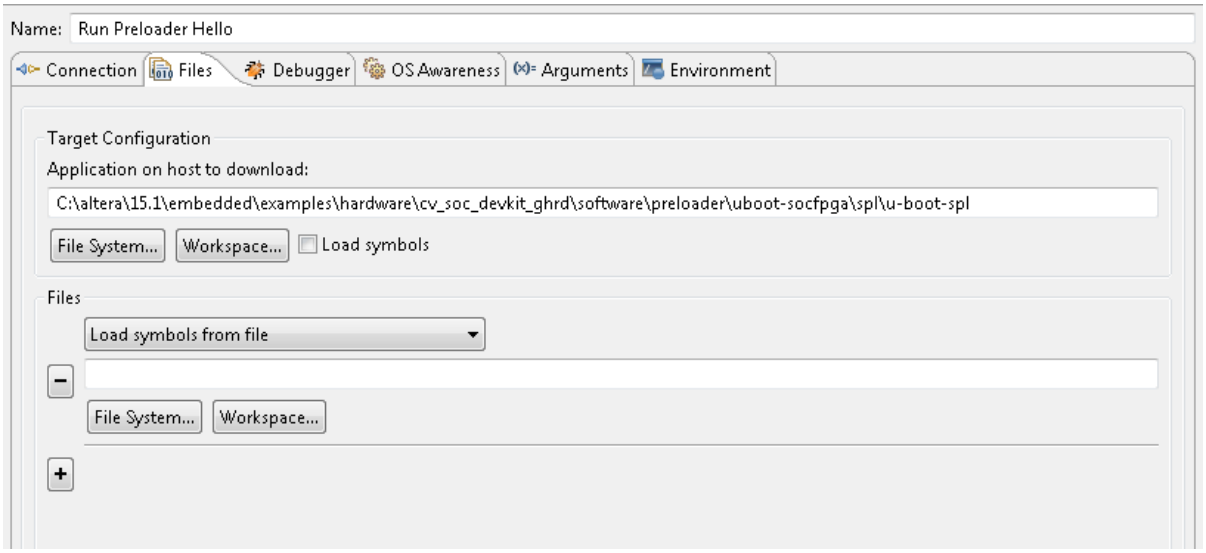
1. Select "USB-Blaster" from the Target Connection pull-down menu and then select an available Bare Metal Debug Connection by clicking **Browse**.

Figure 31: Run Preloader Hello Connection Settings



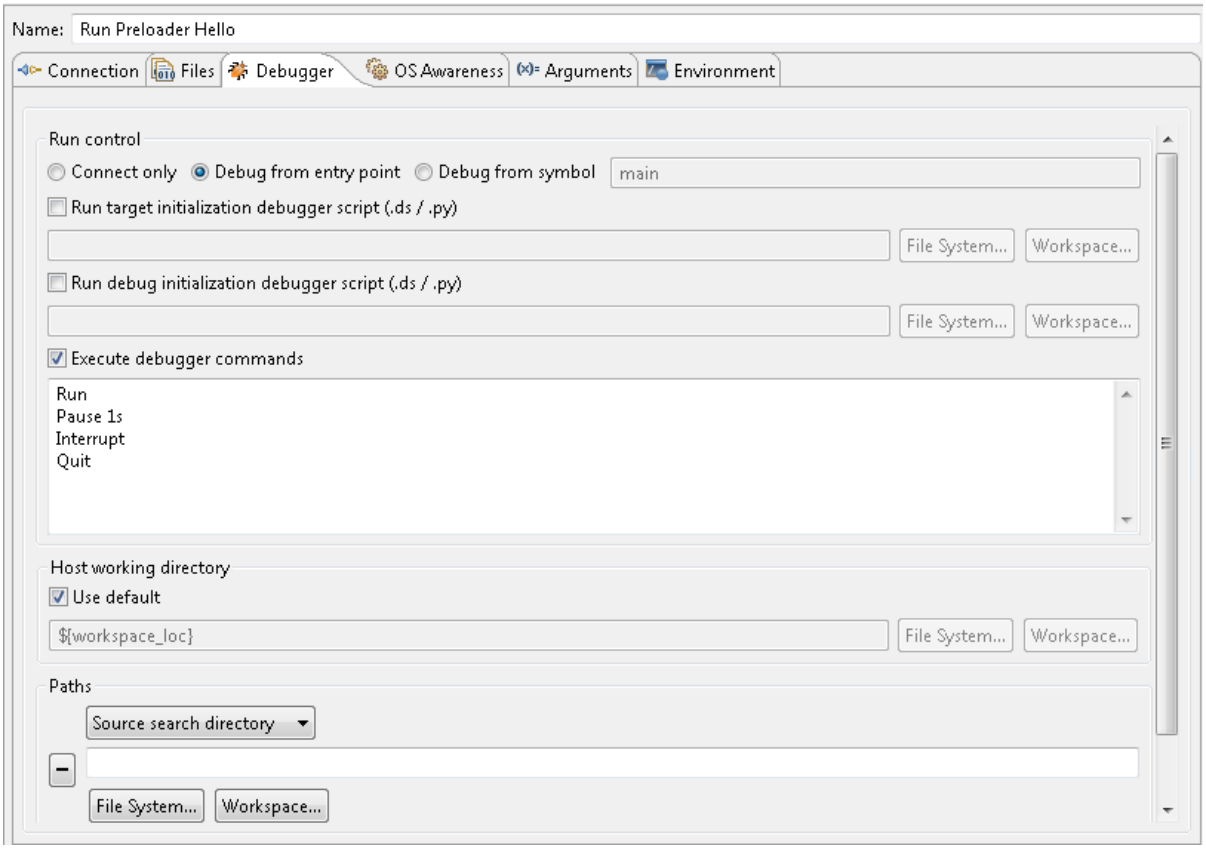
2. On the **Files** tab, select **Workspace** and browse to the project and then the Preloader file "`u-boot-spl`".
3. Uncheck the check box next to "Load Symbols".

Figure 32: Run Preloader Hello Files Settings



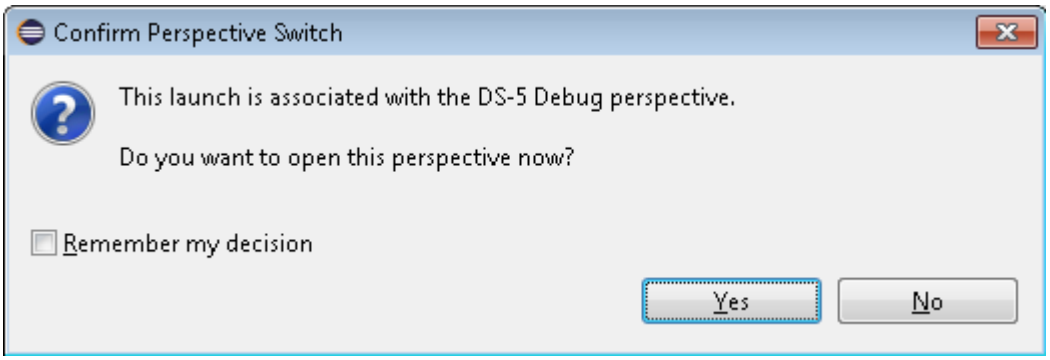
4. On the **Debugger** tab, select "Debug from Entry Point" under **Run Control** and select **Execute debugger commands** to enable the following commands:
   - Run
   - Pause 1s
   - Interrupt
   - Quit

Figure 33: Run Preloader Hello Debugger Settings



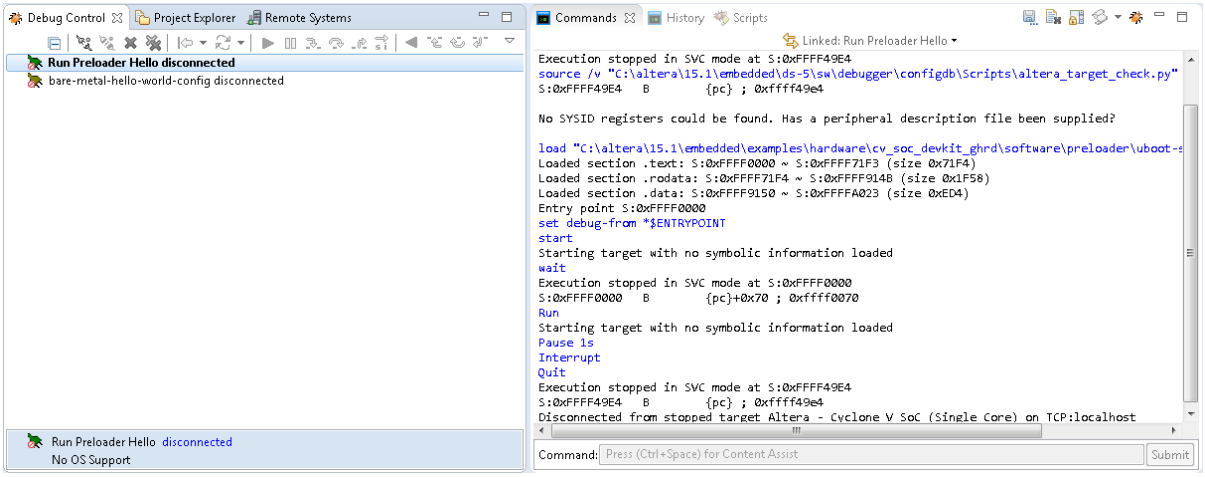
5. Select **Apply** and then **Debug**, then "Yes" to switch to the DS-5 Debug perspective, if queried.

Figure 34: Confirm Perspective Switch



6. DS-5 AE should load and run the Preloader in the on-chip RAM, similarly to how the simple Bare Metal example is ran. This initializes the SDRAM memory controller and then stop and wait. The display should look something like this:

Figure 35: Run Preloader Hello Debug View



This **Run Control** can now be used to launch the Preloader.

Although this method associates the Preloader image with a specific DS-5 project, it can still be used with other projects. However, it is important to remember which project contains the Preloader image.

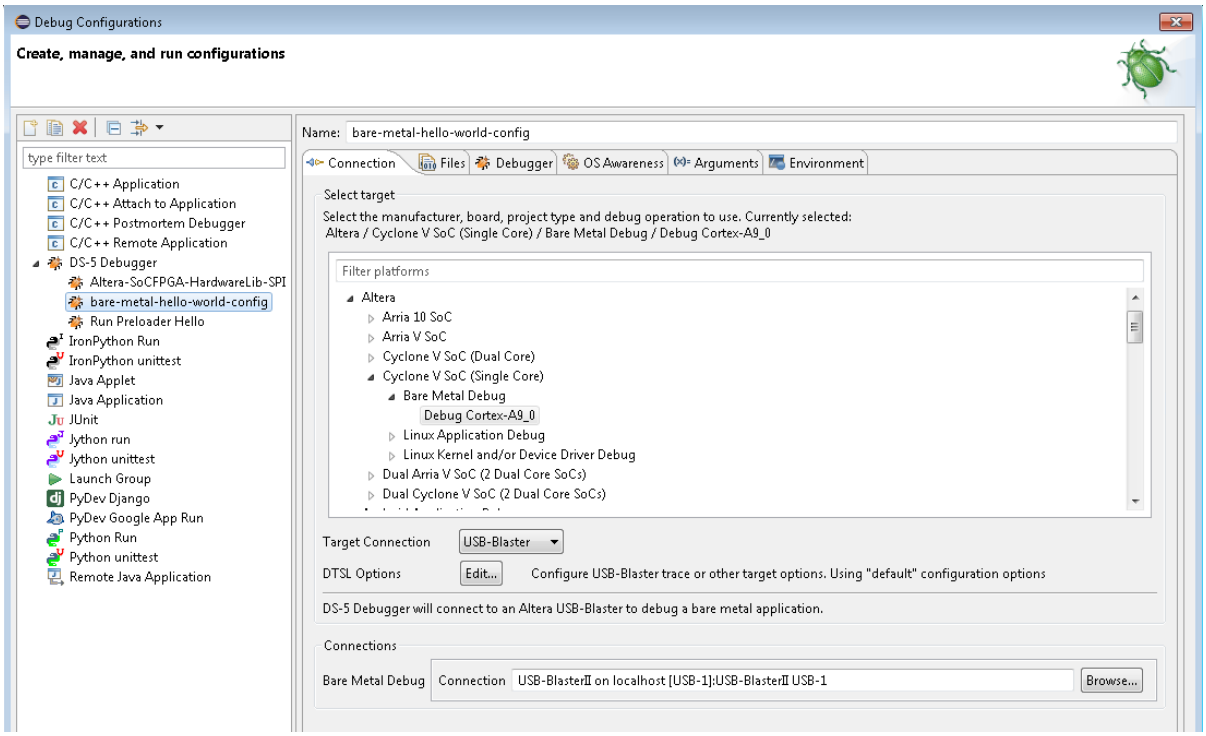
It should load and run the Preloader in the on-chip RAM, similarly to how the simple Bare Metal example was run. This configures the SDRAM memory controller and then stops.

After that, you can download and run the **"Hello World"** example in the SDRAM memory. The same Debug Configuration created earlier for the **Hello World** example can be used.

Download and Debug "Hello World" Example in the SDRAM Memory

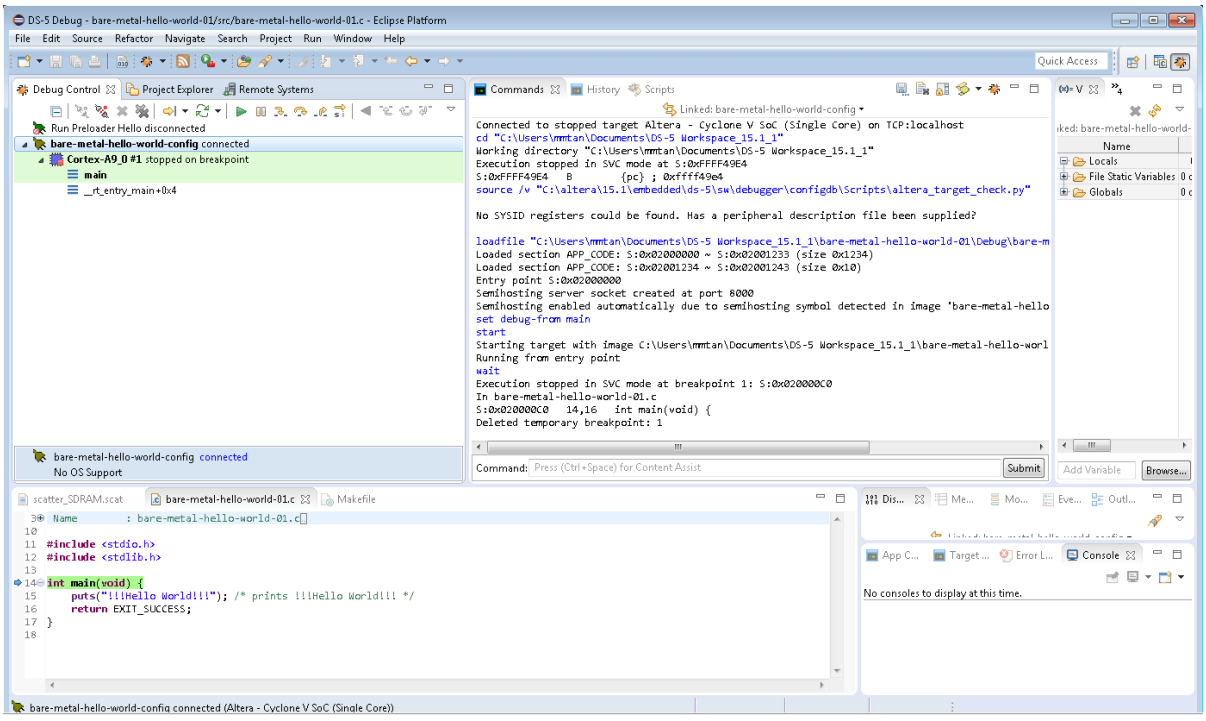
1. Select **Run** > **Debug Configurations**.
   Select the same debug configuration.

Figure 36: Bare Metal "Hello World" Config Debugger



2. Select **Debug** to launch.

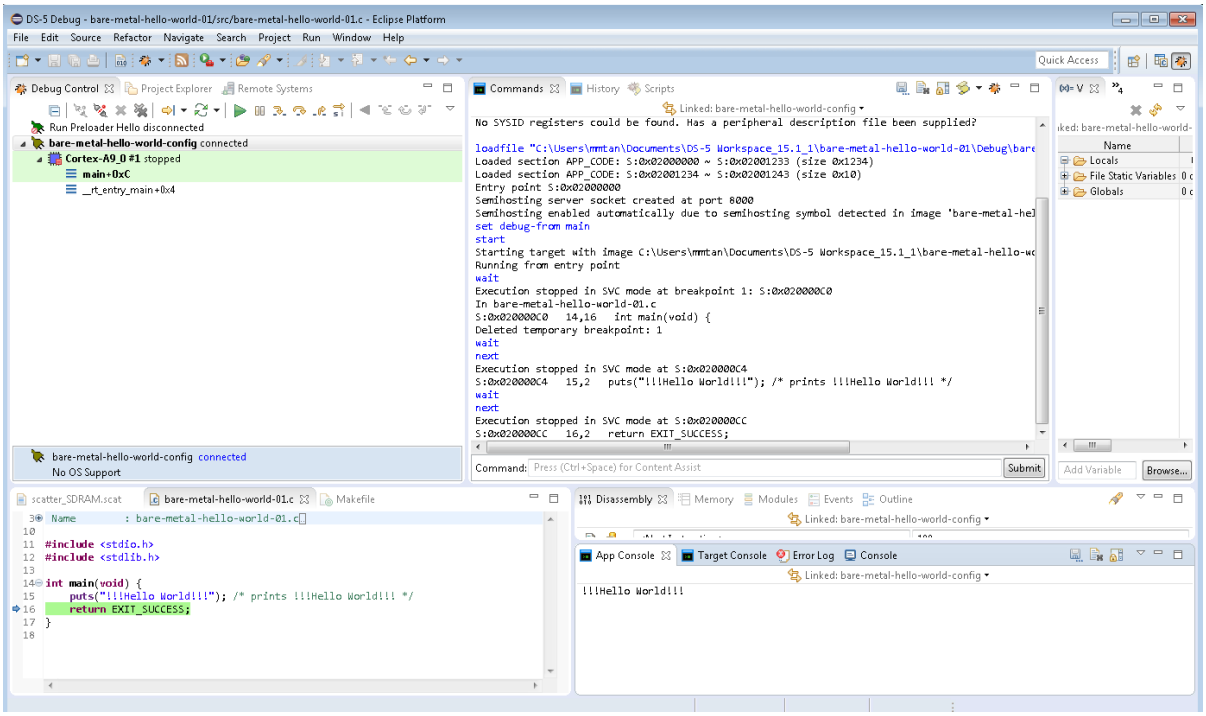
Figure 37: Run "Hello World" from SDRAM Stops at Main



You should notice in the `Commands` view that the entry point is now `S:0x02000000`, which is the beginning of the mapped SDRAM.

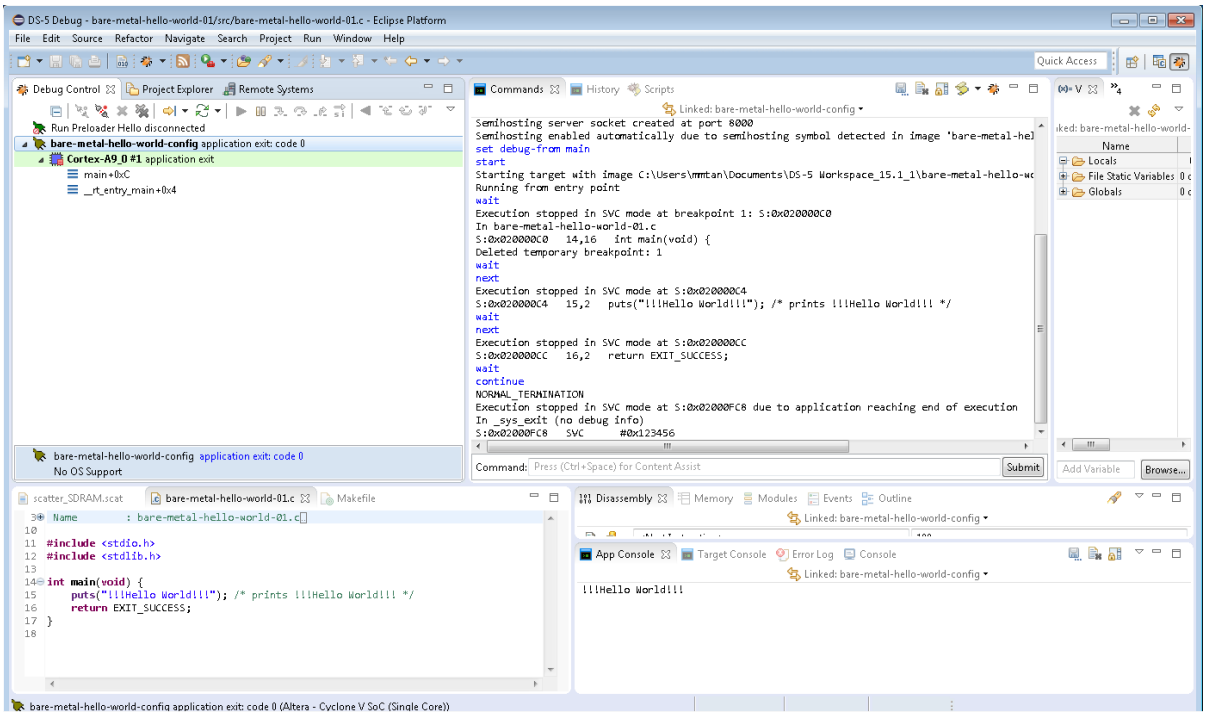
3. Click on the **"Step Over Source Line"** icon or press **F6** to see the program counter progress to the next source line.
4. Click it again to see the "!!!Hello World!!!" message in the `App Console` view.
   If the `App Console` view is not currently selected, then the letters are highlighted in bold letters to indicate that there is a message. Select the `App Console` view to see the output.

Figure 38: Bare Metal "Hello World" App Console



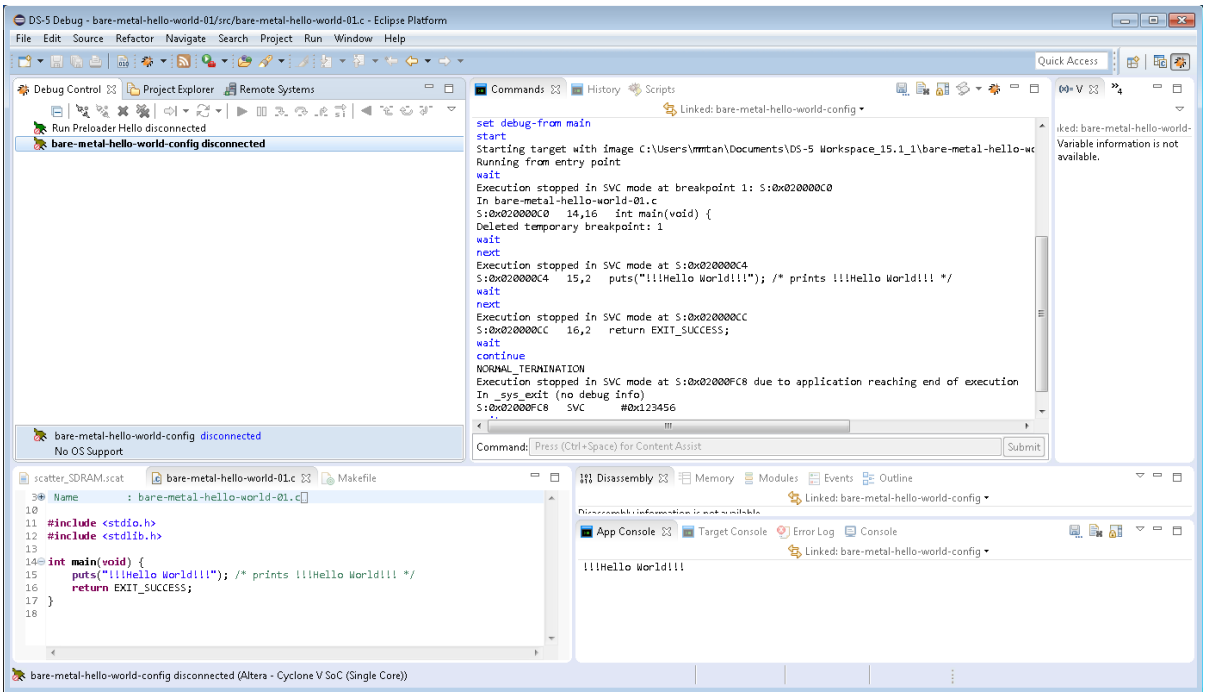
5. Select **Continue** to finish executing the program.

Figure 39: Bare Metal "Hello World" App Console End



6. To rerun, select the debug connection from the list in the `Debug Control` view. In this case, it is "bare-metal-hello-world-01"; and click the yellow arrow.
   This runs the same debug configuration again (and run from Main).
7. When you are finished with this application, you can disconnect it from the target. To disconnect, right click on the **Debug Connection** and then click on the "Disconnect from Target".

Figure 40: Bare Metal "Hello World" App Console Disconnected



It is not necessary to remove the disconnected debug connection from the `Debug Control` view; however, it can be removed since it only needs to run once.

Note: If you do not remove a disconnected debug connection, then you can re-launch the configuration by selecting it from the `Debug Control` view and clicking the **Connect to Target** icon above it.

Don't forget that if you reboot the board, then you have to run the Preloader to configure the memory controller before running an application in SDRAM. One quick way to do this is to keep the disconnected debug connection in the `Debug Control` view; then select it and click on "Connect to Target". This reproduces the previous launch.

8. When you are completely finished with this application, you can remove it from the `Debug Control` view. The connections must be disconnected in order to remove them. To disconnect, select the **Debug Connection** and then click on the "Disconnect from Target". Once disconnected you can select "Remove Connection" or "Remove All Connections".
9. After removing the Debug Connection from the `Debug Control` view, rerun the Preloader by using the Debug Configuration that was created for the Preloader. Rerun the demo, using the Debug Configuration that was created for the **"Hello World"** application.

Modify Project to Run from SDRAM Instead of On-Chip RAM

After completing the process of creating a simple **"Hello World"** application and downloading and debugging it in on-chip RAM of the Altera SoC, the next step is to configure the same project to run in SDRAM instead of on-chip RAM.

To use any SDRAM, the SDRAM controller needs to be configured. This is done by loading and running the Preloader.

For more information about loading and running the Preloader, refer to the "Preloader" section.

Related Information

Preloader on page 20

Create a New Scatter File to Locate the Bare Metal Application in the SDRAM

In the DS-5 ARM compiler projects, it is the scatter file that specifies the addresses that are used to locate the code in the required portion of the memory map.

In this step, use the same **"Hello World"** application, but build it located in the SDRAM memory instead of the on-chip RAM.

To use SDRAM, you must run the Preloader to configure the SDRAM memory controller. If the SDRAM is not configured before you start running the project from SDRAM, the following error messages appears:

```
ERROR(CMD16-TAD274-NAL22):
! Failed to load "bare metal-hello-world-01.axf"
! Failed to write 4,896 bytes to address S:0x02000000 while writing block of 4,096
bytes to address S:0x02000000
! General error on memory or register access.
```

1. Create a scatter file for SDRAM. Go to **File** > **New** > **Others...** and select **Scatter File Editor** > **Scatter File**.
2. Click Next and enter the file name of the new scatter file as **scatter_SDRAM.scat** and click Finish.
3. In the scatter_SDRAM.scat editor view, enter the following:

```
SDRAM 0x02000000 0x02000000  ; 32M SDRAM
{
    APP_CODE  + 0
    {
        * (+ RO , + RW , + ZI )
    }
    ARM_LIB_STACKHEAP 0x03000000 EMPTY 0x0x01000000 ; Application heap and
stack
    { }
}
```

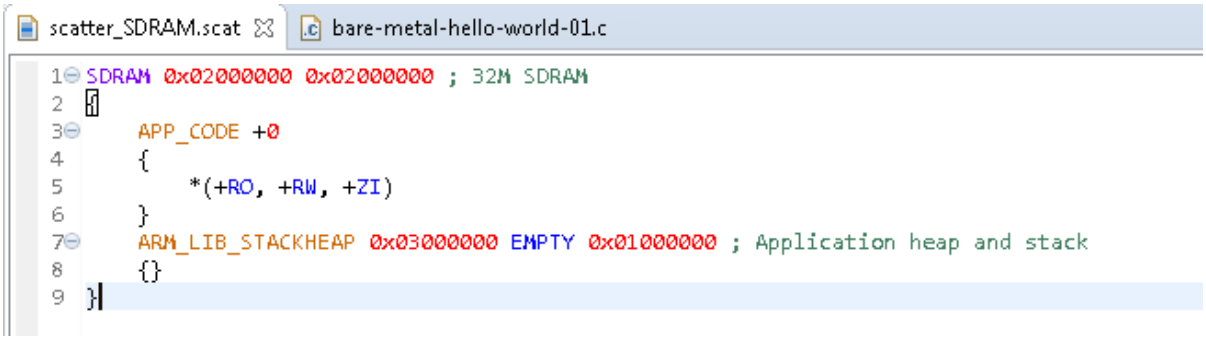
Note: If your HWLibs project needs interrupt support, you have to add a "VECTOR" section at the beginning of the **scatter** file. For example:

```
SDRAM 0x00100000 0x40000000
{
    VECTORS +0
    {
        * (VECTORS, +FIRST)
    }
    APP_CODE +0
    {
        * (+RO, +RW, +ZI)
    }
    ARM_LIB_STACKHEAP +0 EMPTY (0x40000000 - ImageLimit(APP_CODE)) ;
Application heap and stack
    { }
}
```

This is the standard scatter file for a project that runs from SDRAM in most of the HWLibs examples in the "<*SoC EDS installation path*>\embedded\examples\software". Please note that you have to compile and link "**alt_interrupt_armcc.s**" in your project. This is needed because the ARMCC toolchain does not provide the vectors at the start of the program automatically.

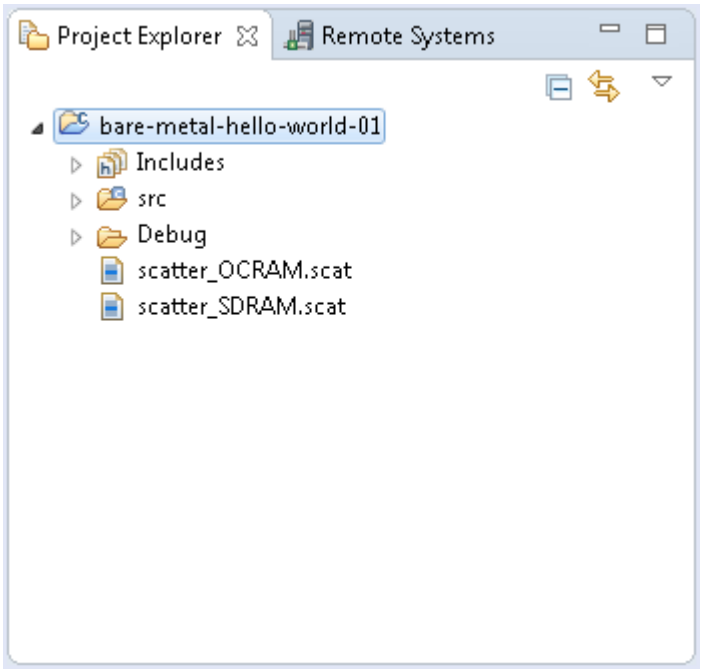
Note: For this section, the Vector section is not included since we are running a simple "**Hello World**" project only.

Figure 41: SDRAM Scatter File Code Snippet



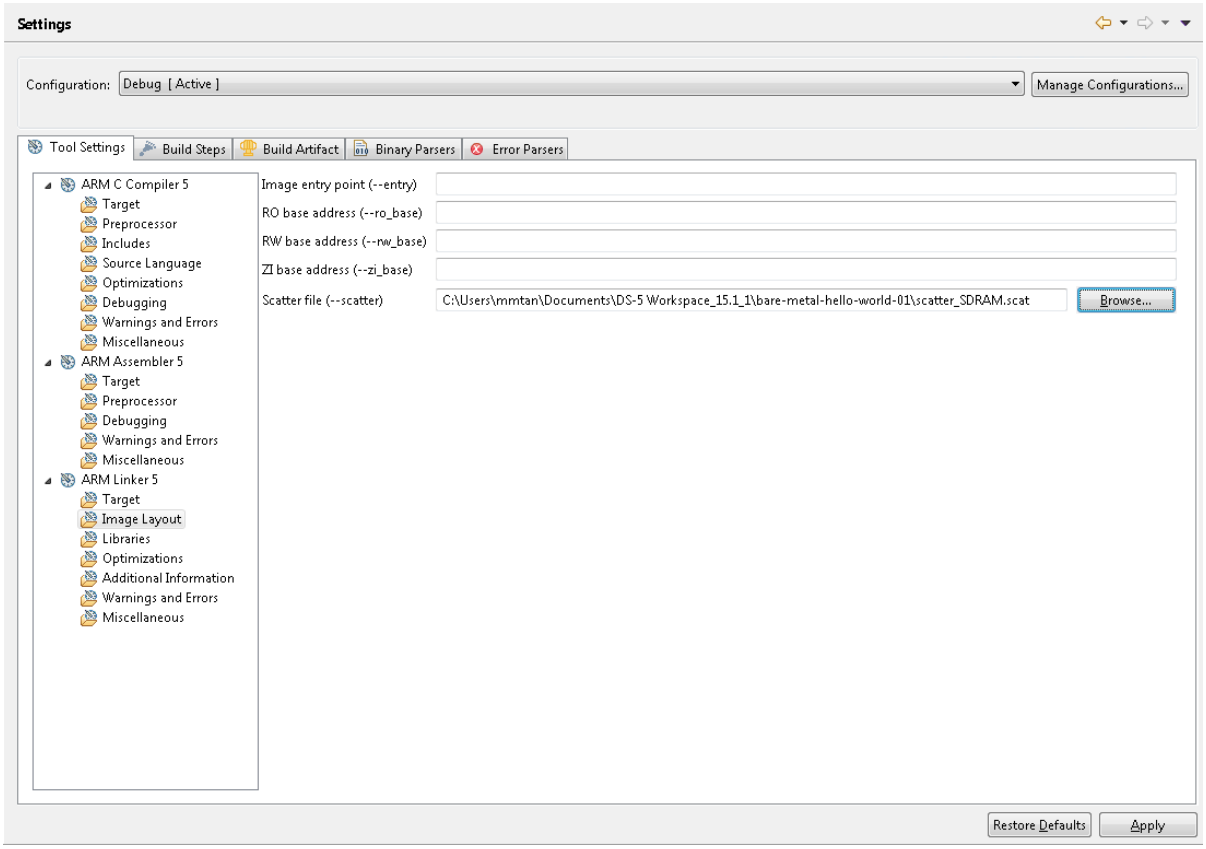
4. Make sure the new scatter file appears in the Project Explorer view.

Figure 42: SDRAM Scatter file in Project Explorer



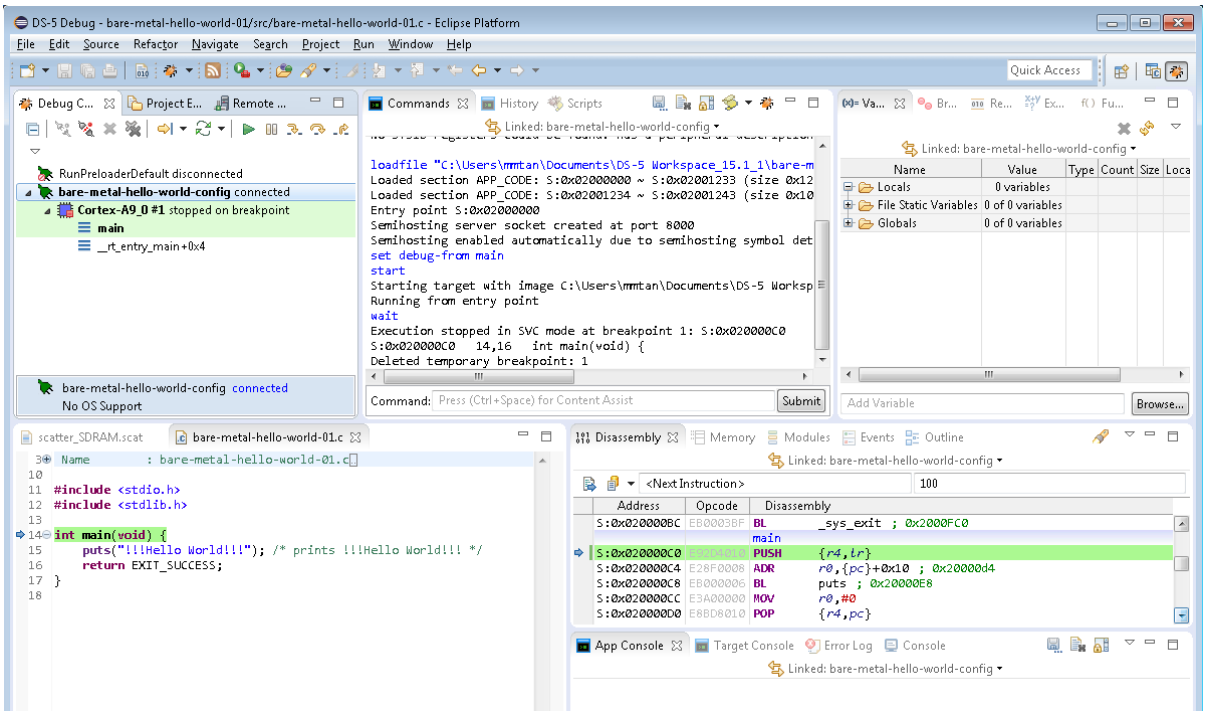
5. Associate the new scatter file with the project. Right-click on the project and select **Properties** > **C/C++ Build** > **Settings** > **ARM Linker 5** > **Image Layout**.
6. Browse to the new scatter file.
7. Alternatively, instead of browsing, enter a path, like: `${workspace_loc}\bare-metal-hello-world-01\scatter_SDRAM.scat`); select **Apply** and then **OK**.

Figure 43: SDRAM Scatter File Location Setting



8. Right-click on the project and select **Build Project** to rebuild with the new scatter file. Use the same debug configuration created earlier to download and run the application.
   The program stops at main and waits there. In the **Commands** view, notice the entry point is `0x02000000`, which is the start address specified in the scatter file for the SDRAM.

Figure 44: Program Stops at SDRAM Start Address



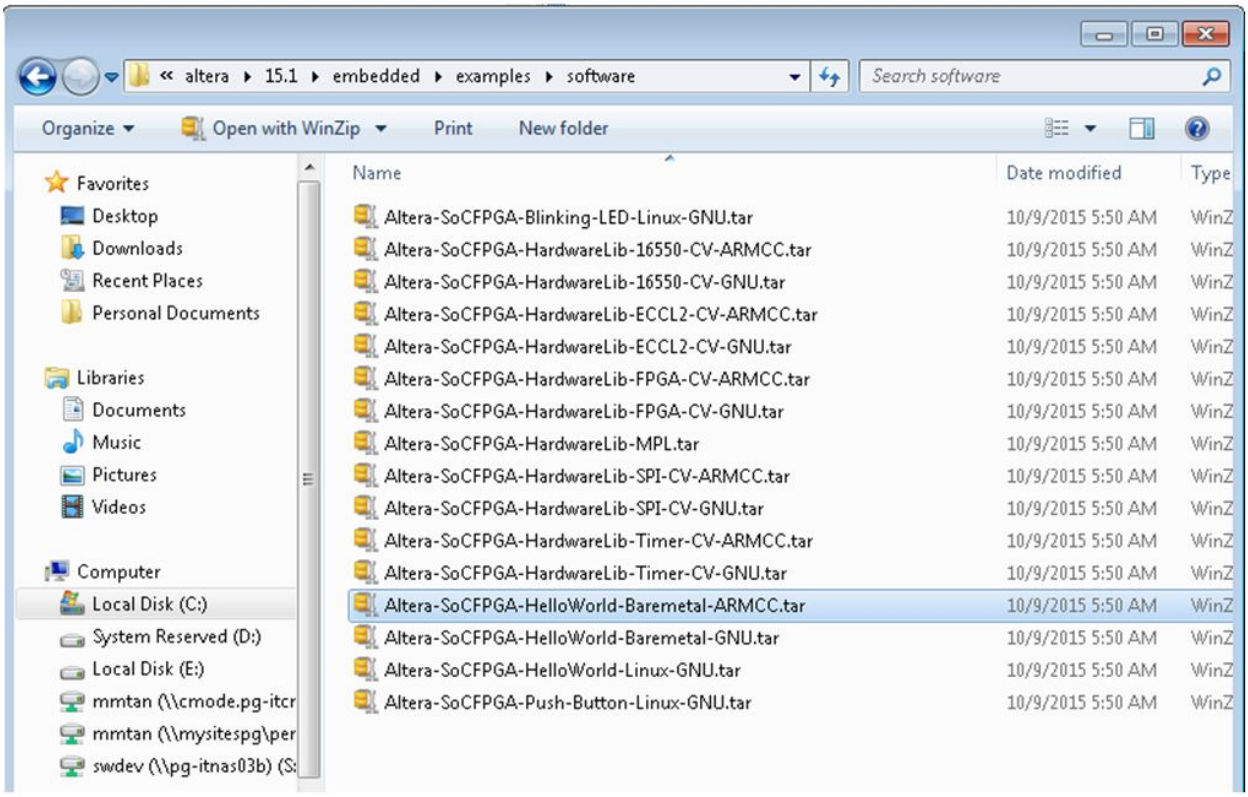
Alternative Way to Create a Simple Bare Metal Project Running from On-Chip-RAM

To create a simple bare metal project that runs from OCRAM, you can also choose to import the Bare Metal example that is included with the Altera SoC EDS tools into the ARM DS-5 environment. For the simple **"Hello World"** bare metal project, you can import either one of the following from the "*<SoC EDS installation path>*\embedded\examples\software":

- **Altera-SoCFPGA-HelloWorld-Baremetal-ARMCC.tar** (using ARM compiler)
- **Altera-SoCFPGA-HelloWorld-Baremetal-GNU.tar** (using GNU CC compiler)

Note: The above bare metal project examples can run on Arria 10, Arria V or Cyclone V SoC Development Kits.

Figure 45: Embedded Software Example Design List



These examples are **make-based** examples where the projects compile based on the Makefile settings or configurations. Creating a simple Bare Metal project manually as shown in the "Simple Bare-Metal Project Using On-Chip-RAM" section is creating a **managed-make** project where the **makefile** is auto-generated.

For more information on how to import, build and debug the project, you can refer to "Importing, Building and Debugging in a Make-Based Example" section.

Related Information

- **Importing, Building and Debugging in a Make-Based Example** on page 40
- **Simple Bare Metal Project Using On-Chip-RAM** on page 5

Importing, Building and Debugging in a Make-Based Example

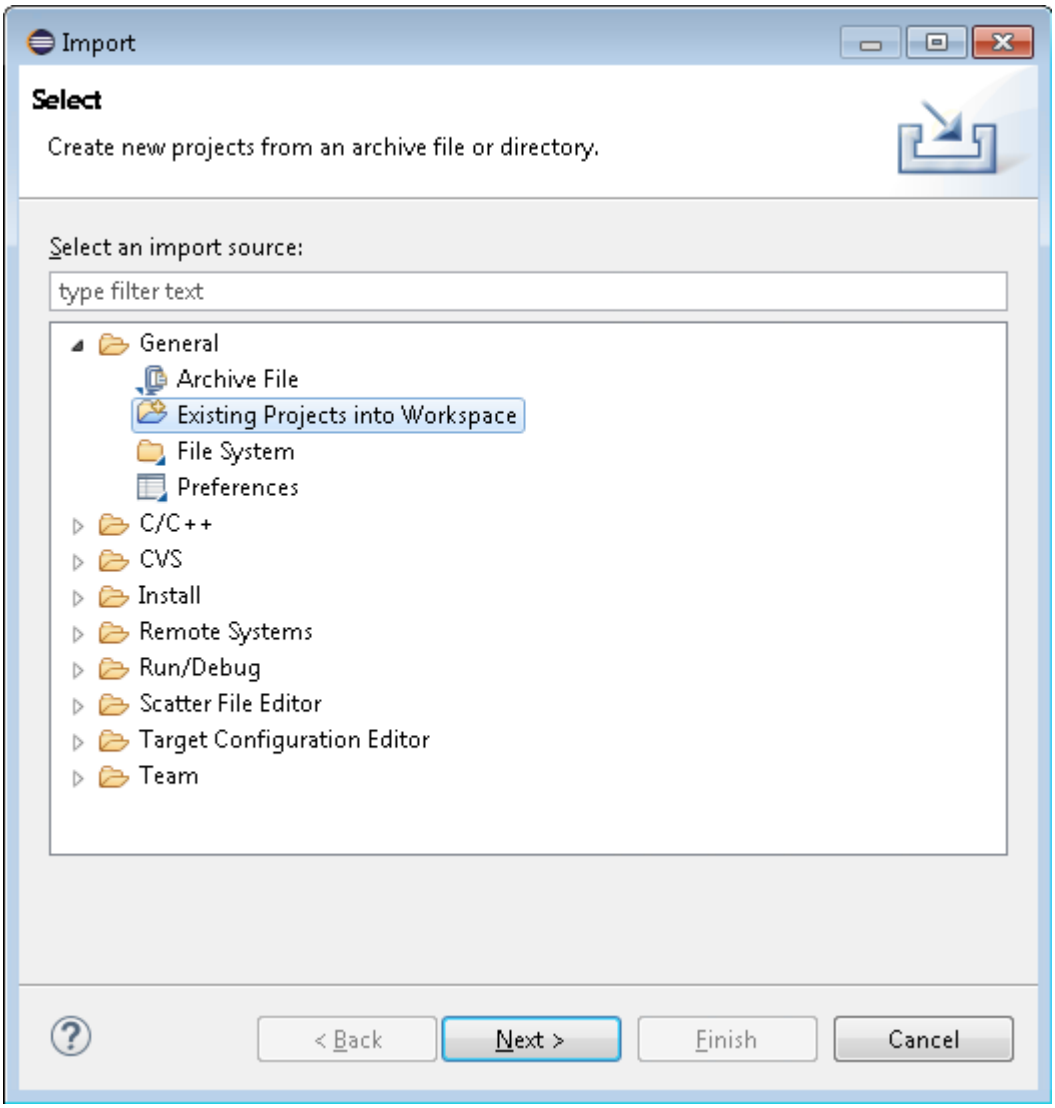
The import, build and debug of the Altera SoC EDS Make-based example covers importing and debugging an example included with the Altera SoC EDS tools into the ARM DS-5 environment.

The Make-based example that is included in the following sections, loads the Golden Hardware Reference Design (GHRD) FPGA image on Cyclone V SoC Development Kit and blinks LEDs to test that the HPS can control peripherals in the FPGA fabric. This example compiles using the ARM compiler.

Import the Project

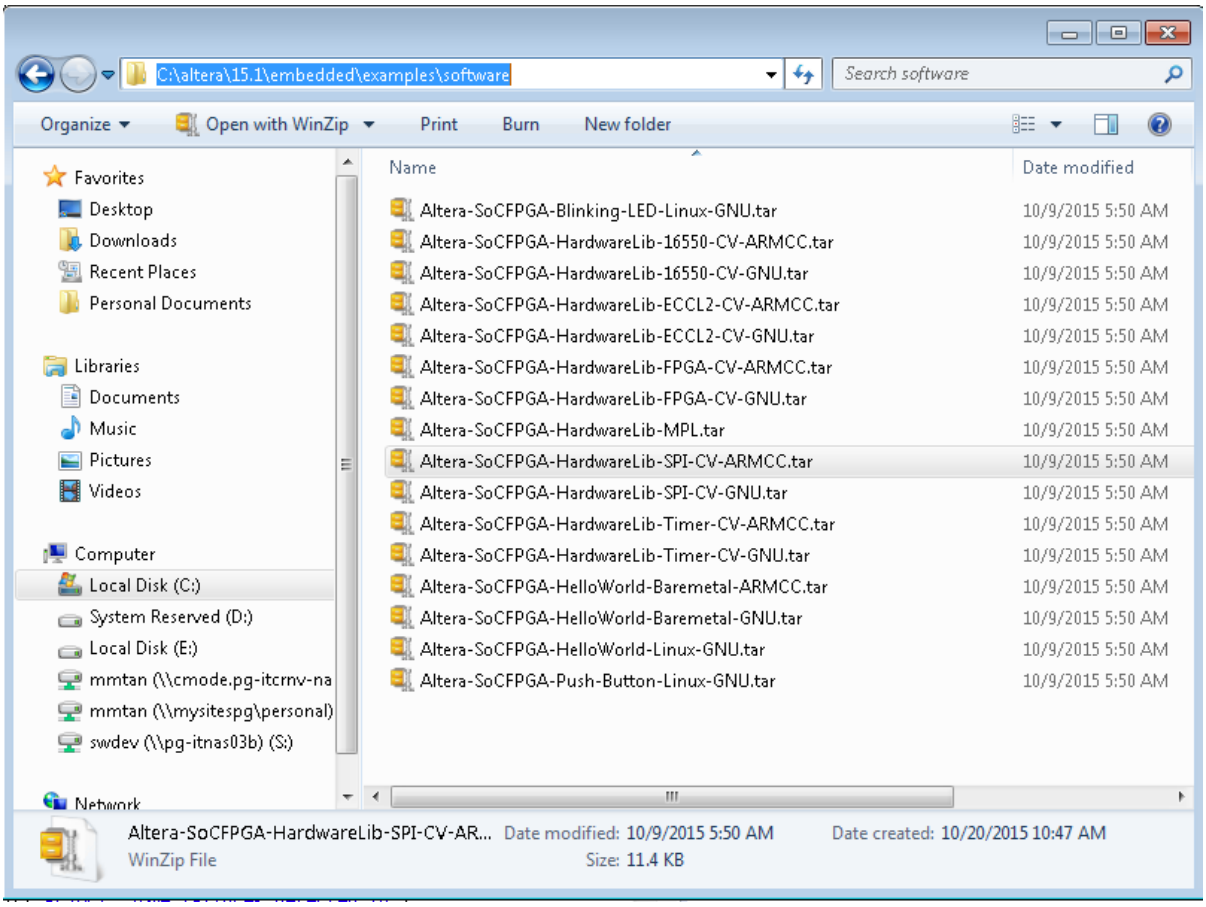
1. From within **DS-5,** select **File** > **Import...**
2. In the resulting dialog box, select "Existing Projects into Workspace" and click **Next**.

Figure 46: Import Existing Projects



3. Select "Select archive file" and browse to the software examples directory of your installation, as shown.

Figure 47: Embedded Software Example Design List



Note: The file can be found in <*SoC EDS installation path*>`\embedded\examples\software`. You can import other examples for reference according to your usage.

4. Choose the **"Altera-SoCFPGA-HardwareLib-FPGA-CV-ARMCC.tar.gz"** archive and select **Open**.
5. Click **Finish** to complete the import process.

Build the Project

Now that the project is imported, make sure the current toolchain (ARM Compiler 5 (DS-5 built-in)) is selected correctly in the Tool Chain Editor. This can be done with the following steps:

1. Right click on the project and select "Properties".
2. Go to C/C++ Build > Tool Chain Editor.

Next, right click on the project and select "Build Project". This initiates a Make-based build that does the following:

- Copies additional source files from HWLibs into the project.
- Creates an object file of the FPGA image using standard Altera tools and **objcopy**.
- Compiles and links everything into an AXF executable (ELF-compatible).

Note: For details of what is happening, browse the Makefile that is part of the project.

Included with the project are a number of key items that are used to properly initialize the ARM core:

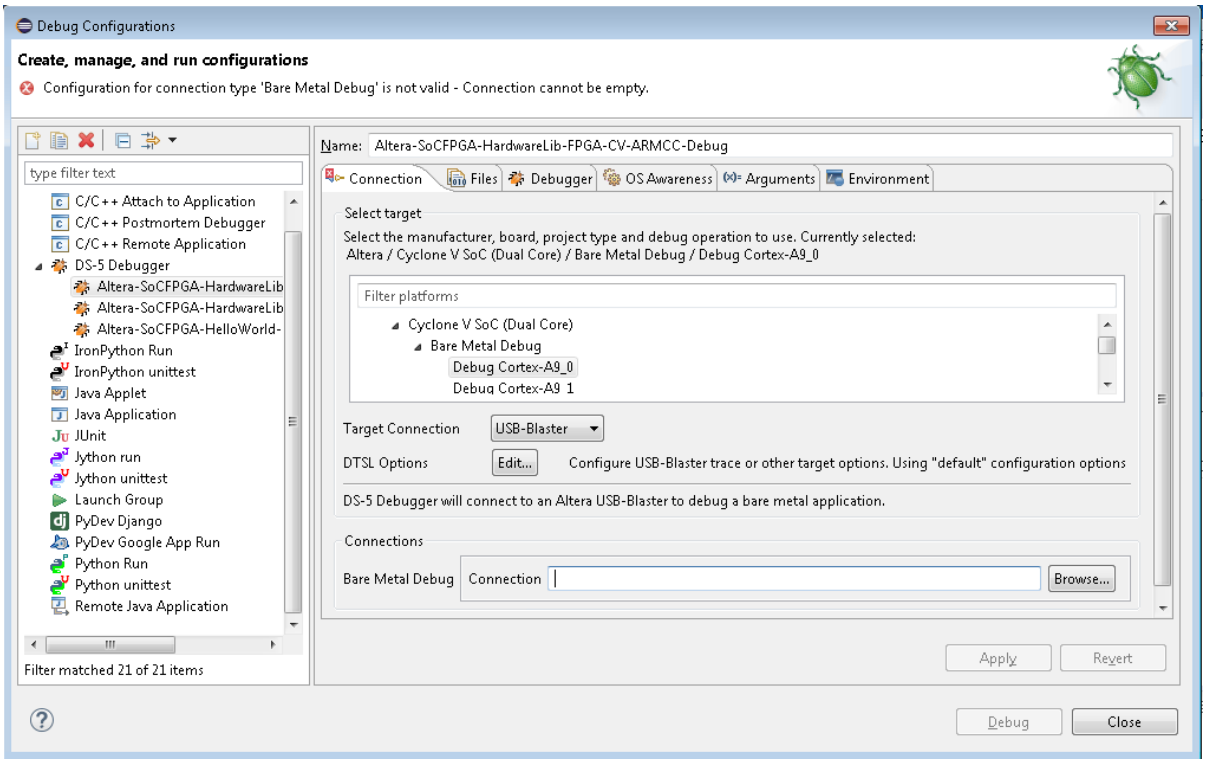
- A scatter file **(scatter.scat)** that the ARM tools use for linking

  Note: Much simpler that GNU linker syntax
- A DS script file **(debug-hosted.ds)** that controls debug flow
  - Loads and runs the Preloader
  - Loads the project executable **(HWLIB.axf)** and stops at the “main” symbol
- A `readme.txt` file which describes more about this example

Debug the Project

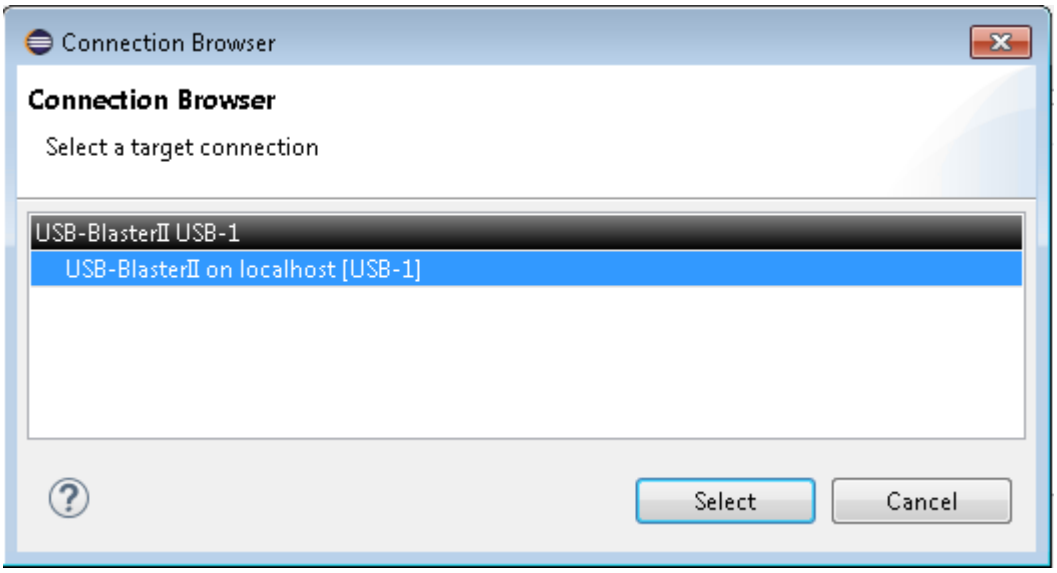
1. Right-click on the project and select **Debug** > **Debug As** > **Debug Configurations**, as shown:

Figure 48: "Altera-SoCFPGA-HardwareLib-FPGA-CV-ARMCC-Debug" Debug Configurations



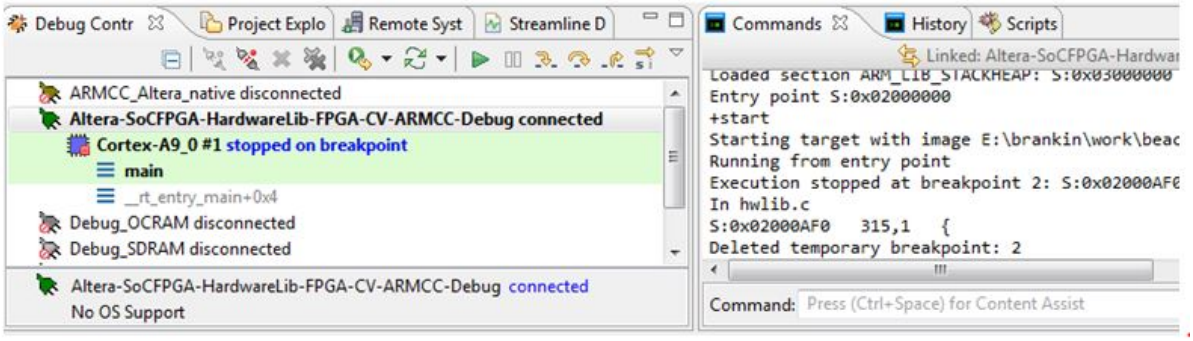
2. Next to the **Connections** field, select **Browse** and select the hardware you are using (Altera USB Blaster or DStream) from the available connections and click **Select**.

Figure 49: Connection Browser



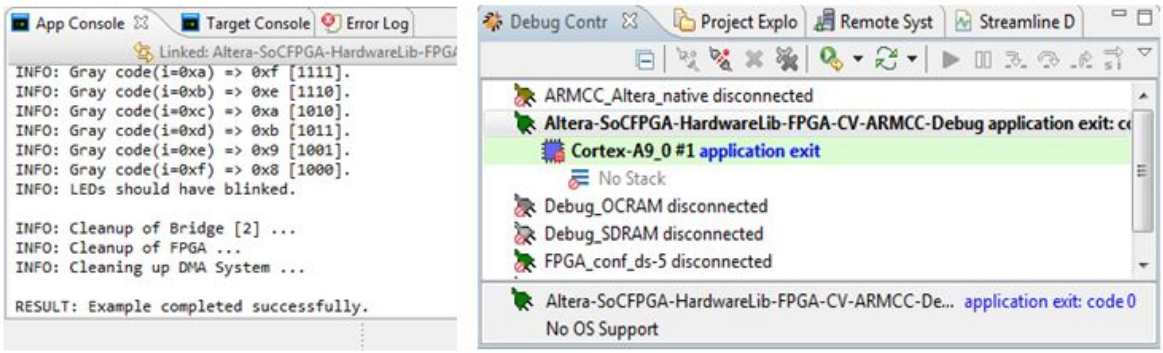
3. Select **Apply** and click **Debug** to start the debug session which will configure the processor, load the software and execute it, stopping at the **"main"** symbol as the default.

Figure 50: Debug Stopping at Main



4. Click the  button to start running the code. The code should run to completion. Other options () are also available on the `Debug Control` window.
The `App Console` window should display correct results.

Figure 51: Application Exit Displayed in App Console



DS-5 ARM HWLIBs Project Derived from Make-Based Project

This section takes a makefile-based example project that comes with SoC EDS and converts it to a managed make project. “Managed Make” in Eclipse means that the IDE (DS-5 in this case) takes care of generating and maintaining any Makefiles.

When you use this method, you are able to change the project settings from the GUI.

Note: You cannot automate generic steps, like converting the FPGA SOF file to C object code to be linked into the application.

A managed make project can be compiled from the command line, since there is a makefile. Therefore, automated builds can be done using commands.

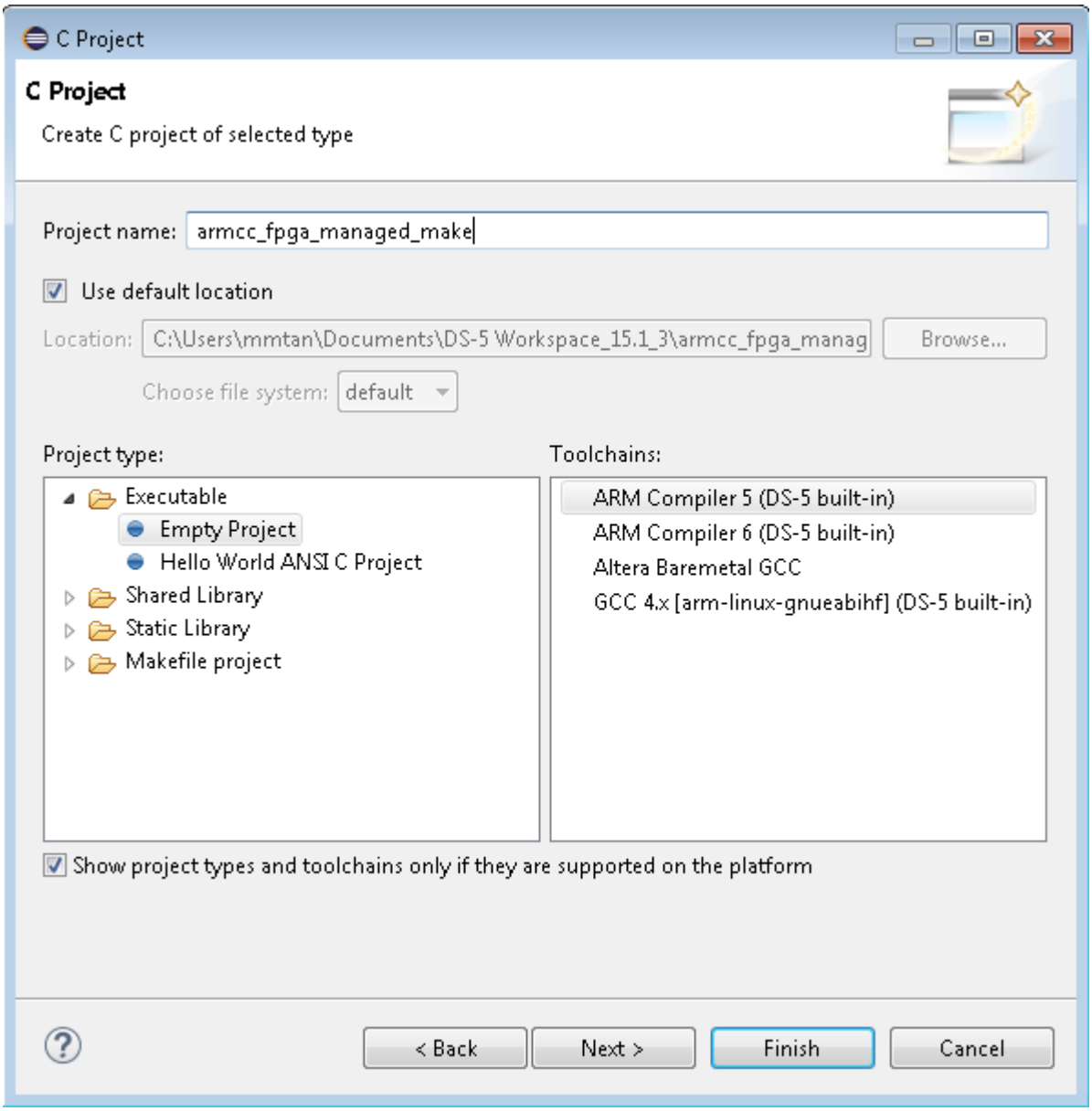
Create Project

Before you begin

Before you start creating a project, make sure you have gone through the "Importing, Building and Debugging in a Make-Based Example" section and the example project compiled successfully.

1. Create a new “managed make” project by selecting **File** > **New** > **C Project**.
2. Fill in the resulting dialog box, as shown, naming it whatever you prefer, for example: **armcc_fpga_managed_make**.

Figure 52: Create New ARMCC FPGA Managed Make Project



Related Information

Importing, Building and Debugging in a Make-Based Example on page 40

Copy Files

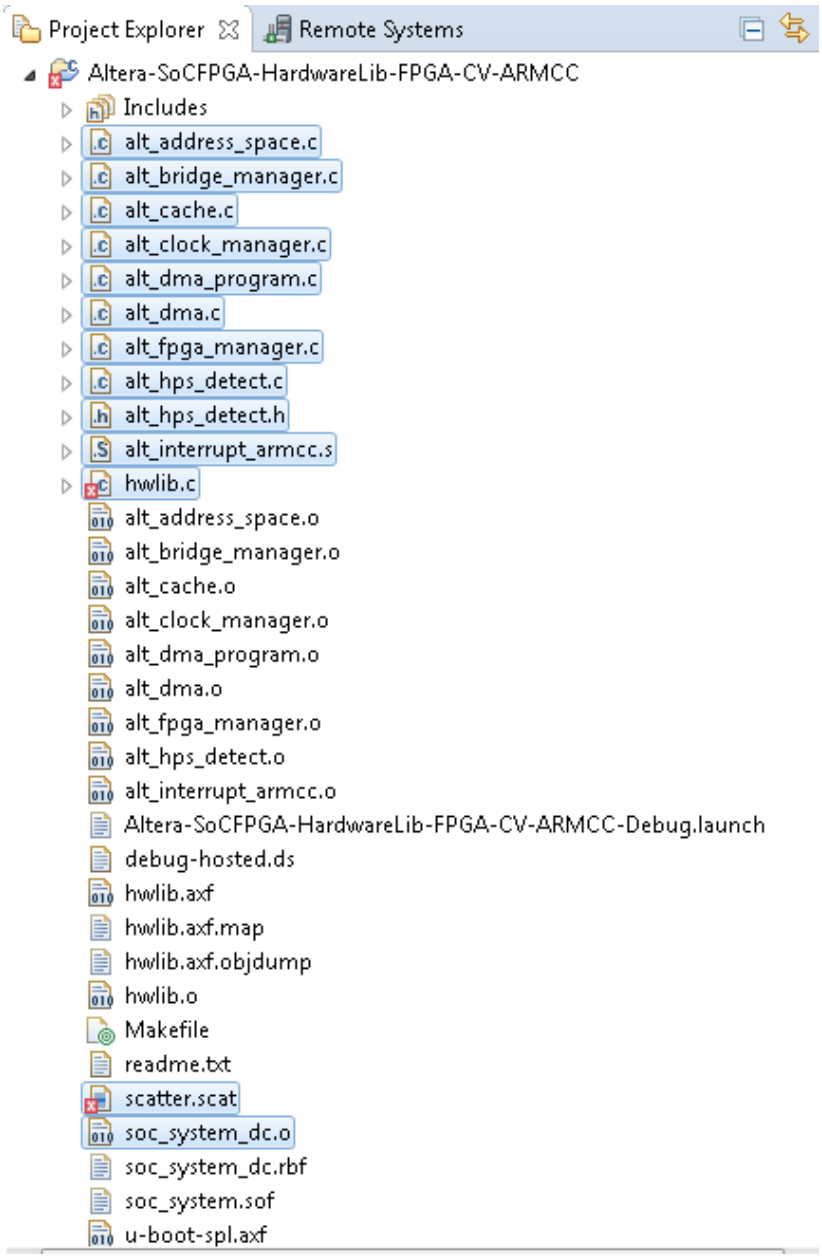
Make sure the HWLibs FPGA ARMCC project has been imported and compiled successfully before you proceed. The compiled project creates the FPGA object code file, previously, auto generated using the makefile.

Before you begin

Copy all C code (files with a **.c** extension), **alt_interrupt_armcc.s**, the scatter file and the FPGA object code **(soc_system_dc.o)** from the Altera SoC EDS example project.

1. Select the files, right-click and click **Copy**.

Figure 53: Files to Copy from the SoC EDS Example Project



2. Right-click the new project **(armcc_fpga_managed_make)** and select **Paste**.
   All of the files shown in the image should have been copied into the new project.

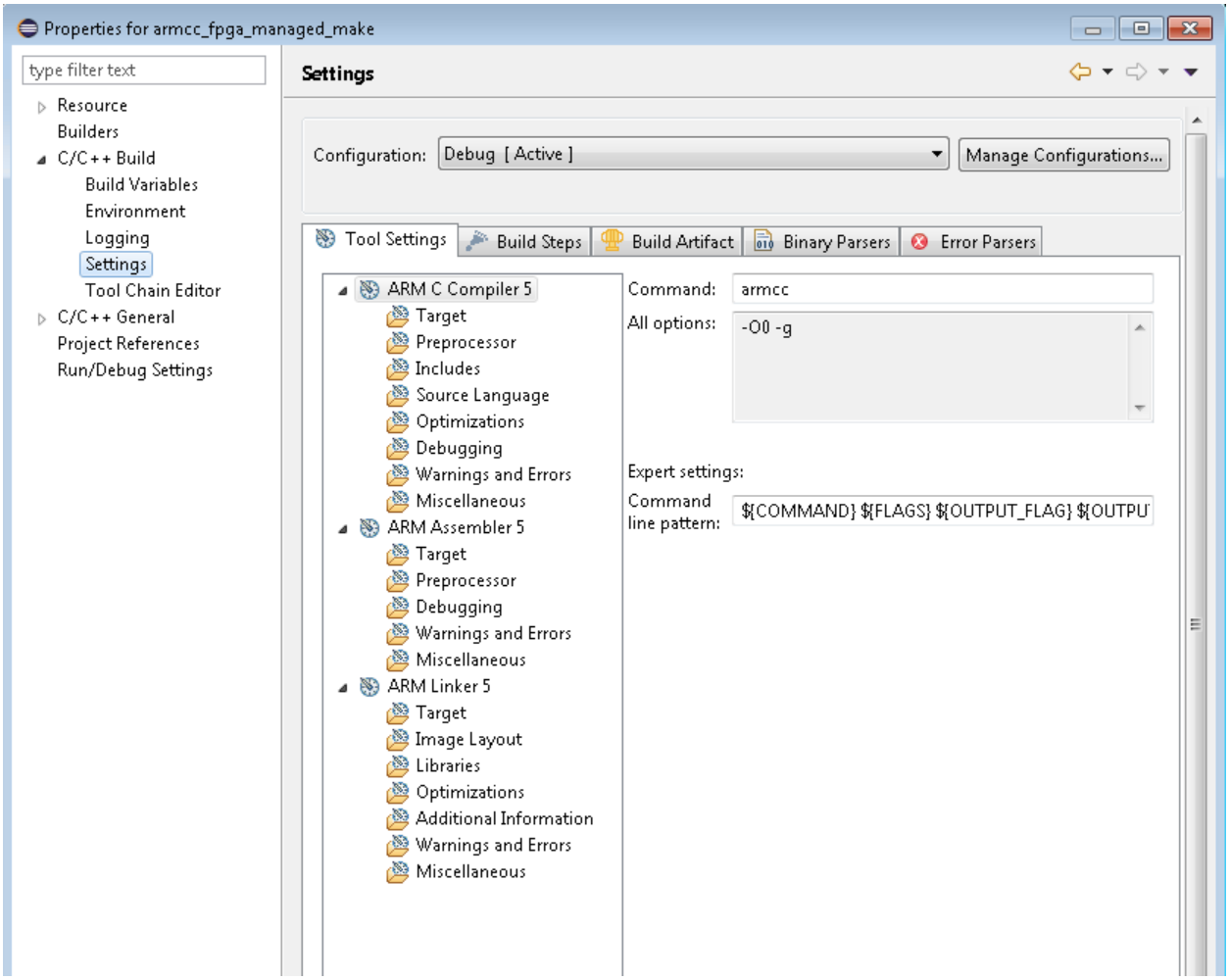
   The C code is built and linked with the FPGA object file, automatically, after setting up the build system in the next section.

Configure Build Settings

Get ready to build.

Right-click on the project in DS-5's `Project Explorer` window and select **Properties** then browse to **C/C++ Build** and select **Settings**.

Figure 54: ARMCC FPGA Managed Make Settings



Compiler, assembler and linker settings changes are necessary. Please follow the steps in the "ARM C Compiler Settings" section.

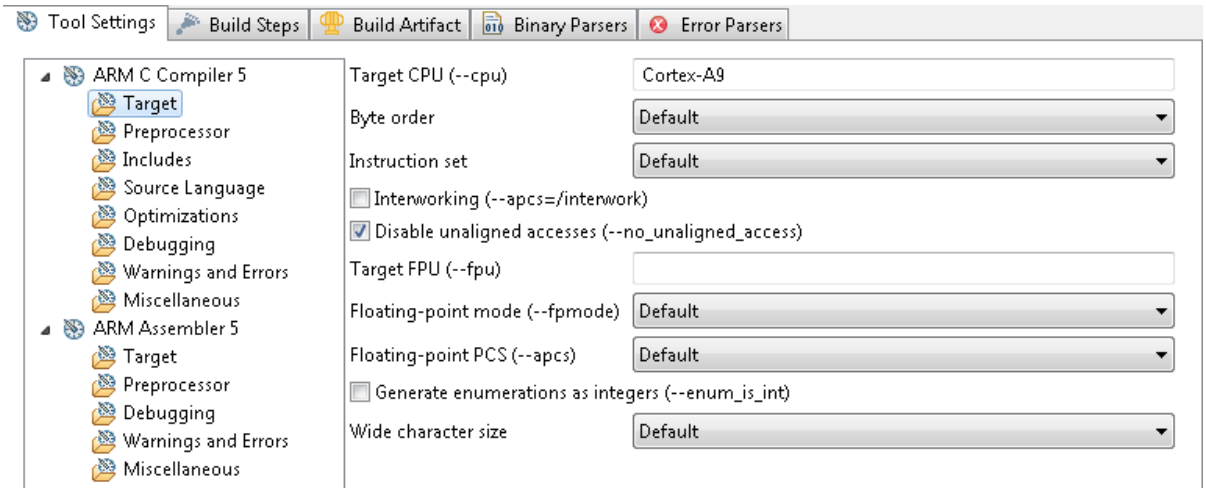
Related Information

ARM C Compiler Settings

ARM C Compiler Settings

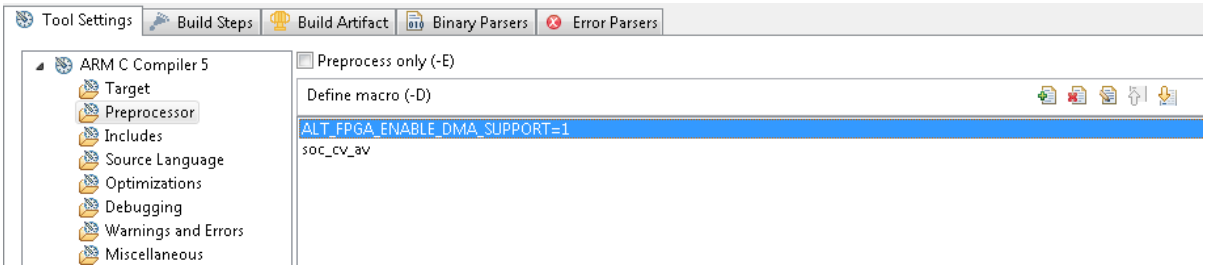
1. In "**Target**", set Target CPU" to **"Cortex-A9"** and check the **"Disable unaligned accesses"** box.

Figure 55: ARMCC Target Settings



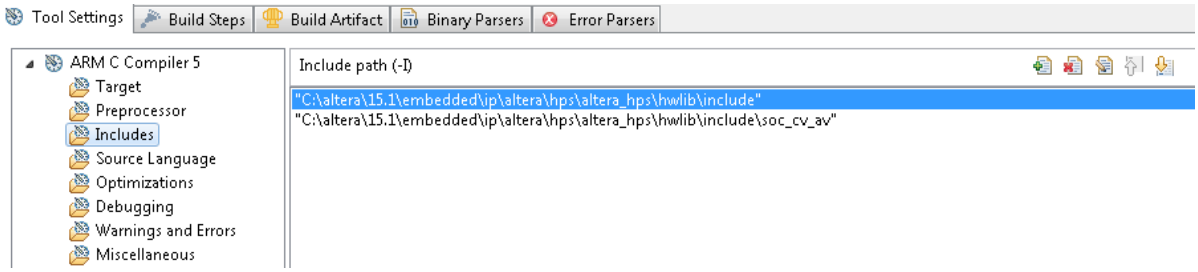
2. Add the preprocessor macro **"ALT_FPGA_ENABLE_DMA_SUPPORT=1"** to ensure that the code responsible for configuring the FPGA uses the DMA in the HPS.

Figure 56: ARMCC Preprocessor Settings



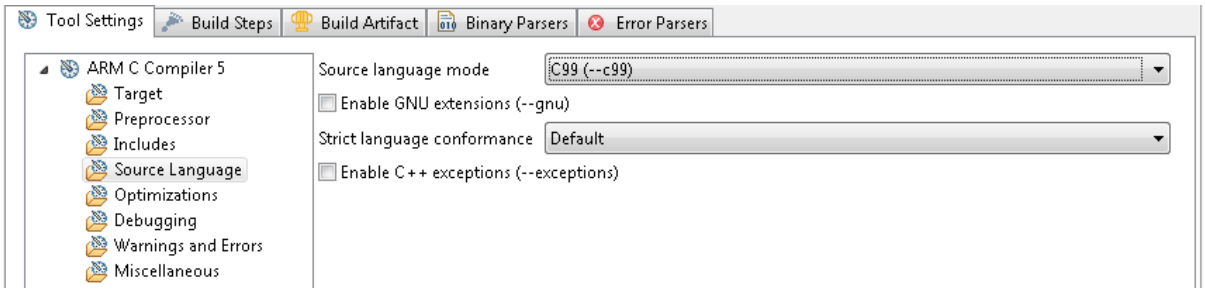
3. Add the preprocessor macro "**soc_cv_av**" to set the Cyclone V device family.
4. Select the next category down (**"Includes"**) and click the **"+"** button to the right to add the include paths to **Build Settings**.

Figure 57: ARMCC Include Path Settings



5. Select "Source Language" and set **"Source language mode"** to "-c99".

Figure 58: ARMCC Source Language Settings

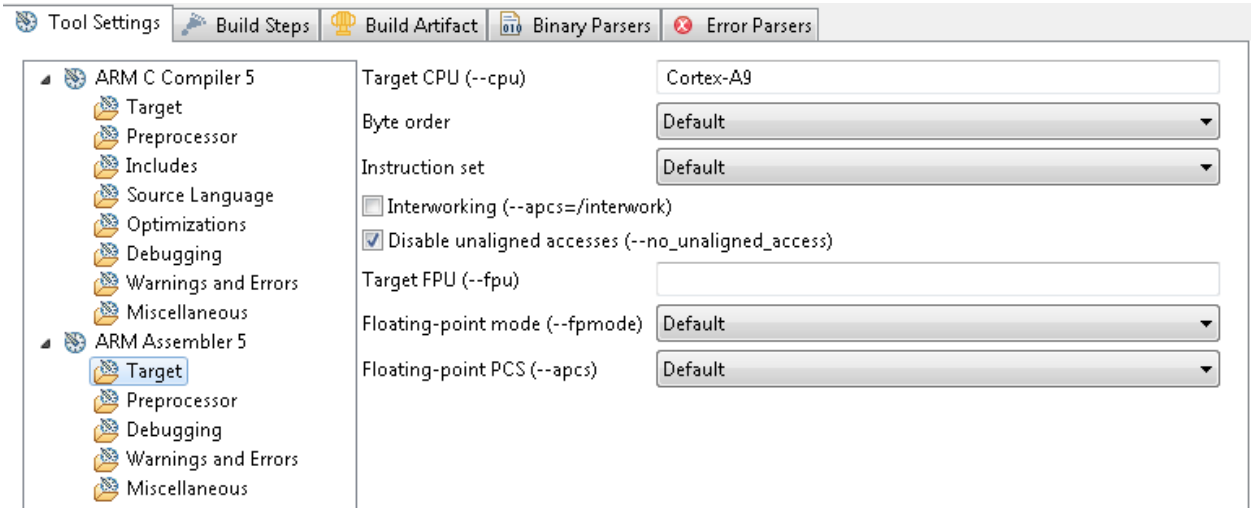


Leave "**Optimizations**", "**Debugging**", "**Warnings and Errors**", and "**Miscellaneous**" settings to default values.

ARM Assembler Settings

Change the settings in "Target" to match the following:

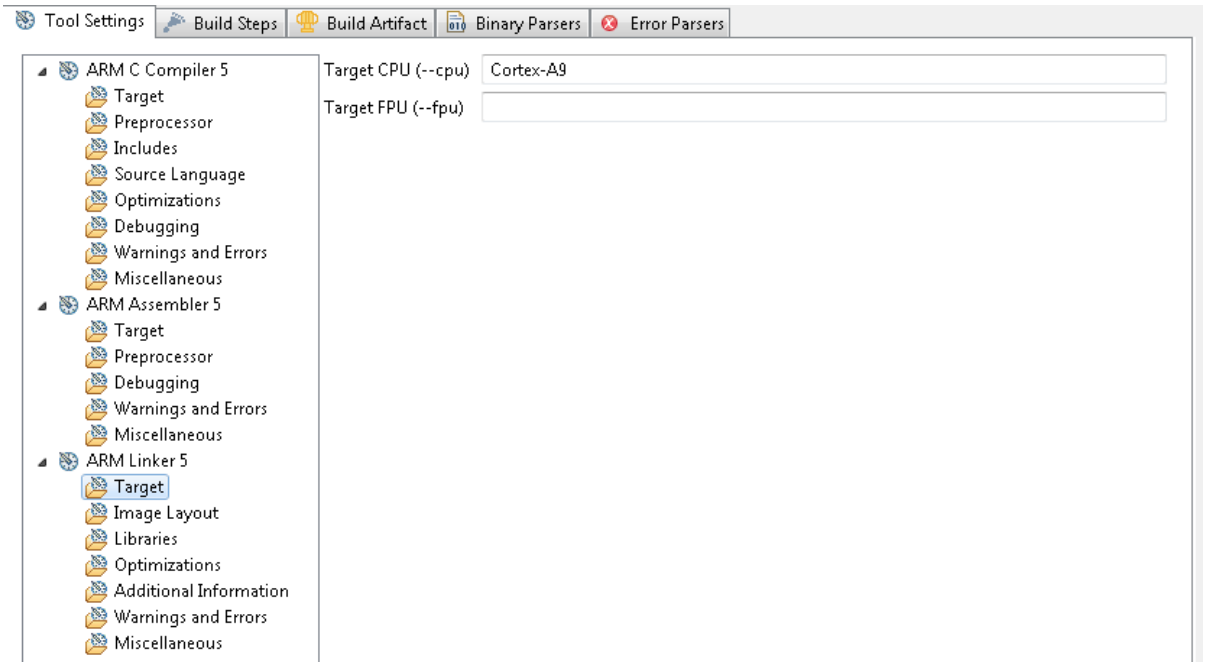
Figure 59: ARM Assembler Target Settings



ARM Linker Settings

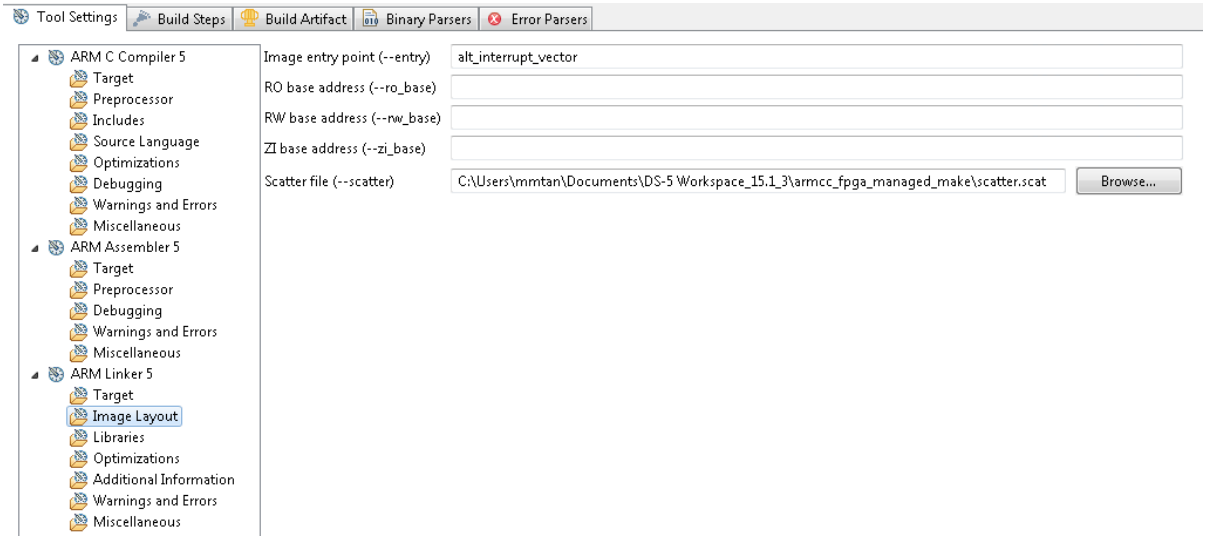
1. Define the "Target CPU (--cpu)" as `"Cortex-A9"` in the **Target** section.

Figure 60: ARM Linker Target Settings



2. "Define the Image entry point (--entry) as "alt_interrupt_vector" and add the Scatter file (--scatter) location in the "**Image Layout**" section.

Figure 61: ARM Linker Image Layout Settings



3. Leave "**Libraries**", "**Optimization**", "**Additional Information**", and "**Warnings and Errors**" with default values.
4. Add the FPGA object file in the **Miscellaneous** settings section.
5. Click on the **Add...** icon and Browse to the location of the FPGA object file under the "armcc_fpga_managed_make" Workspace.

Figure 62: ARM Linker Miscellaneous Settings

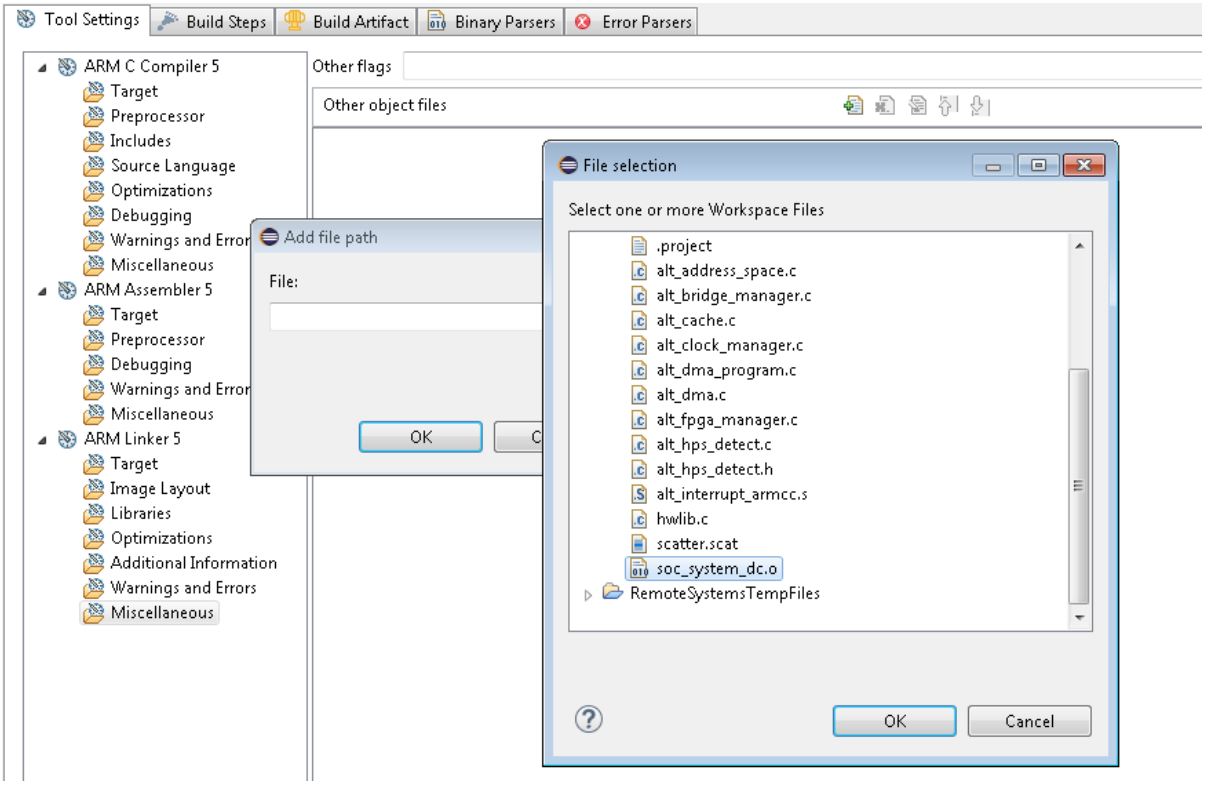
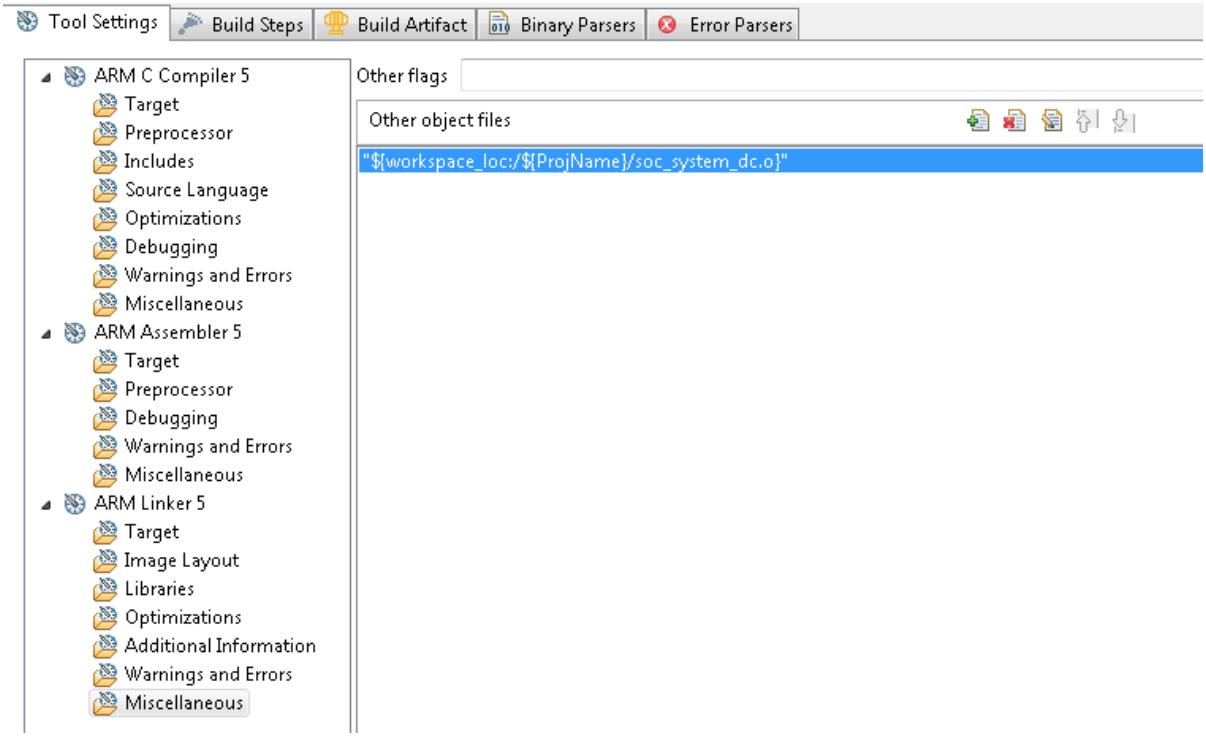


Figure 63: ARM Linker Miscellaneous Settings - Part 2



6. Click **Apply** and then **OK** to apply settings and return.

Build Project

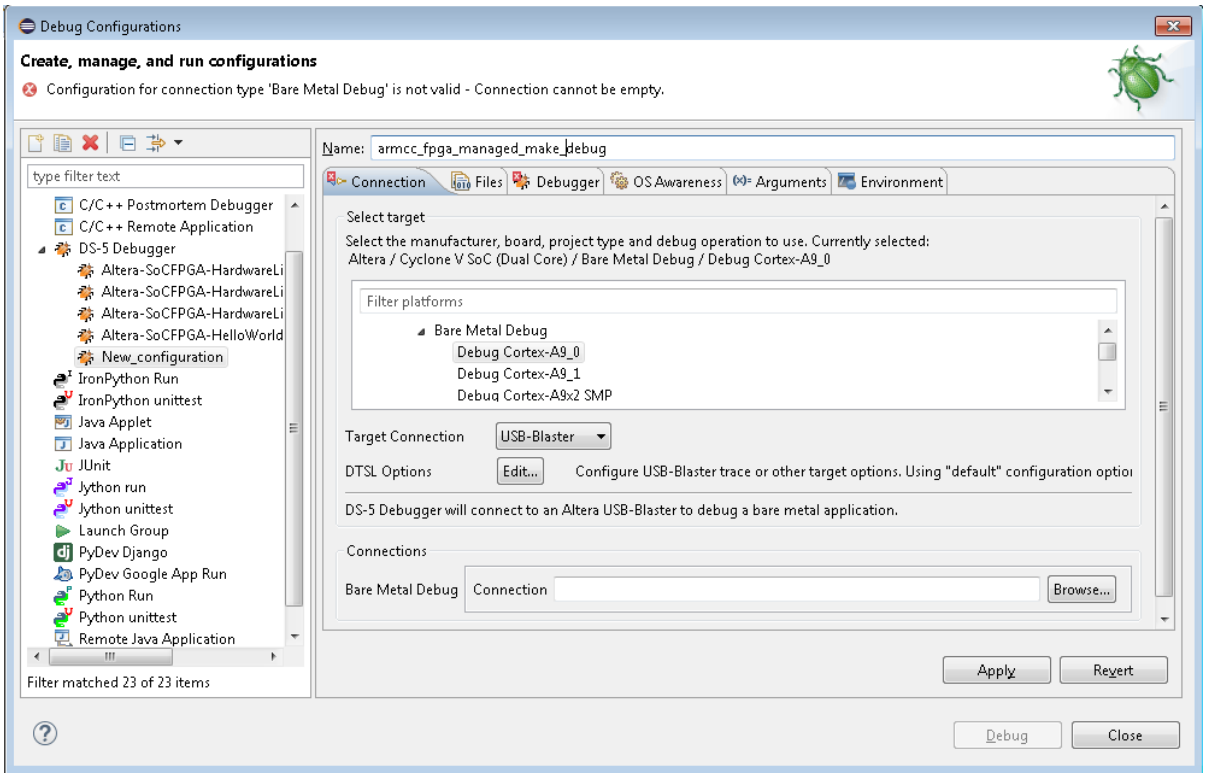
Right-click on the project you created and pull-down to the "Build Project" option. This starts building the project in the default build directory in the project.

Run/Debug Project

Create a Debug Configuration

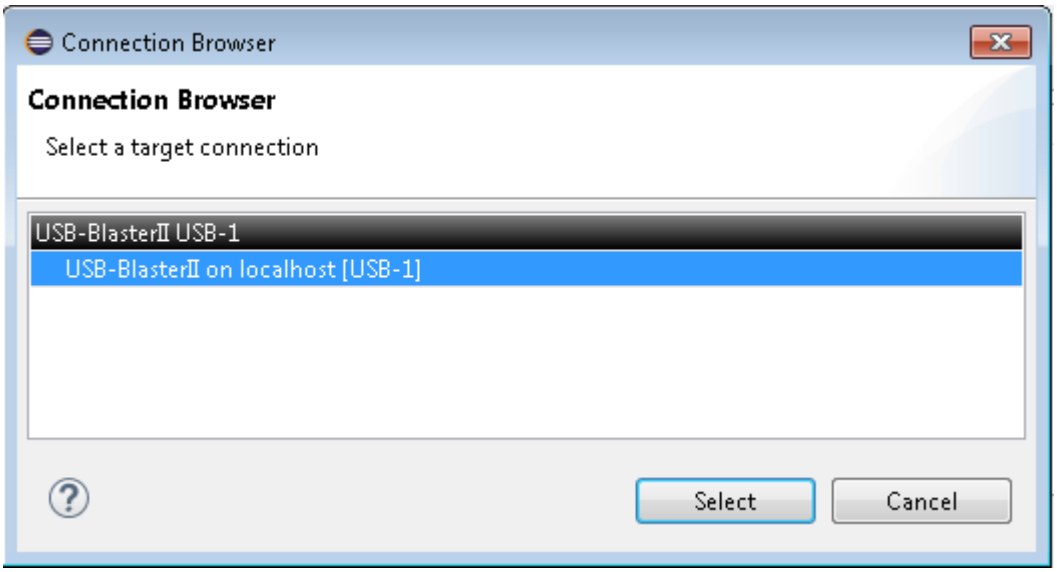
1. Right-click on the project and pull-down to the **Debug As** > **Debug Configurations** option. This opens the `Debug Configurations` dialog box.
2. Create a new configuration and setup the debug hardware as shown.

Figure 64: ARMCC FPGA Managed Make Debug Connection Settings



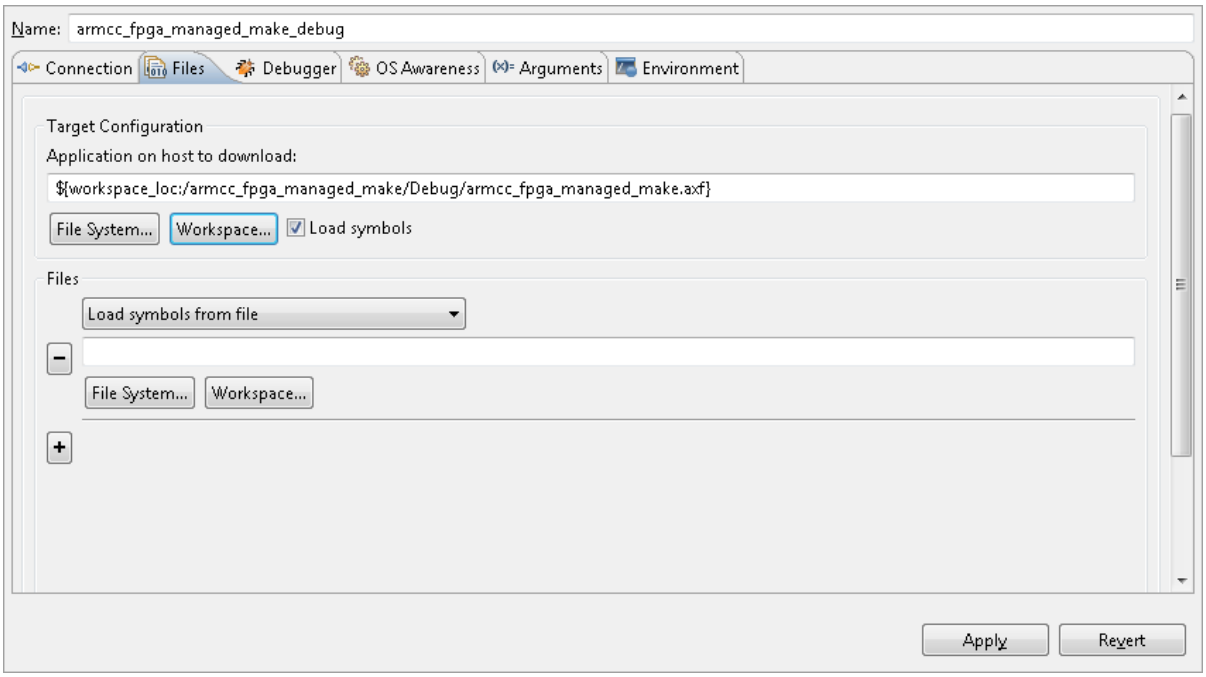
3. Select an available Bare Metal Debug Connection by clicking **Browse**. This returns a list of the available debug connections.

Figure 65: Connection Browser



4. Click **OK** and then go to the **Files** tab on the top right. Select "**Workspace...**" and look for the **AXF** file in the project **Debug** sub-directory.

Figure 66: ARMCC FPGA Managed Make Files Settings



5. Click on **Apply** and then **Debug** to start the debugging session.

Minimal Preloader

The Minimal Preloader (MPL) is an alternative for the General Public License (GPL) Preloader. It uses the BSD license and may be freely distributed and modified according to the terms of that license. The MPL supports a subset of features supported by Altera's GPL Preloader.

The MPL initializes the PLLs, reset signals, configures IOCSR and pin MUXing, and performs other configuration-based on the preloader generator file settings. It can also load the FPGA from a boot source, if desired. It then reads a secondary image from a boot source into RAM and hands control to that image.

This version of MPL supports booting from QSPI, SD/MMC and FPGA.

Note: NAND boot is not supported.

The MPL uses Altera HWLib drivers for most of its functionality. It also uses Altera HWLib SoCAL folders for the memory map definitions and basic read and write commands.

The MPL supports both the ARMCC and GNU GCC compilers. The MPL supports both Cyclone V SoC and Arria V SoC devices.

Note: Arria 10 SoC devices are not supported.

The example project in the following section is for Cyclone V SoC. Please modify the example as needed to select appropriate file names.

Minimal Preloader Example Project

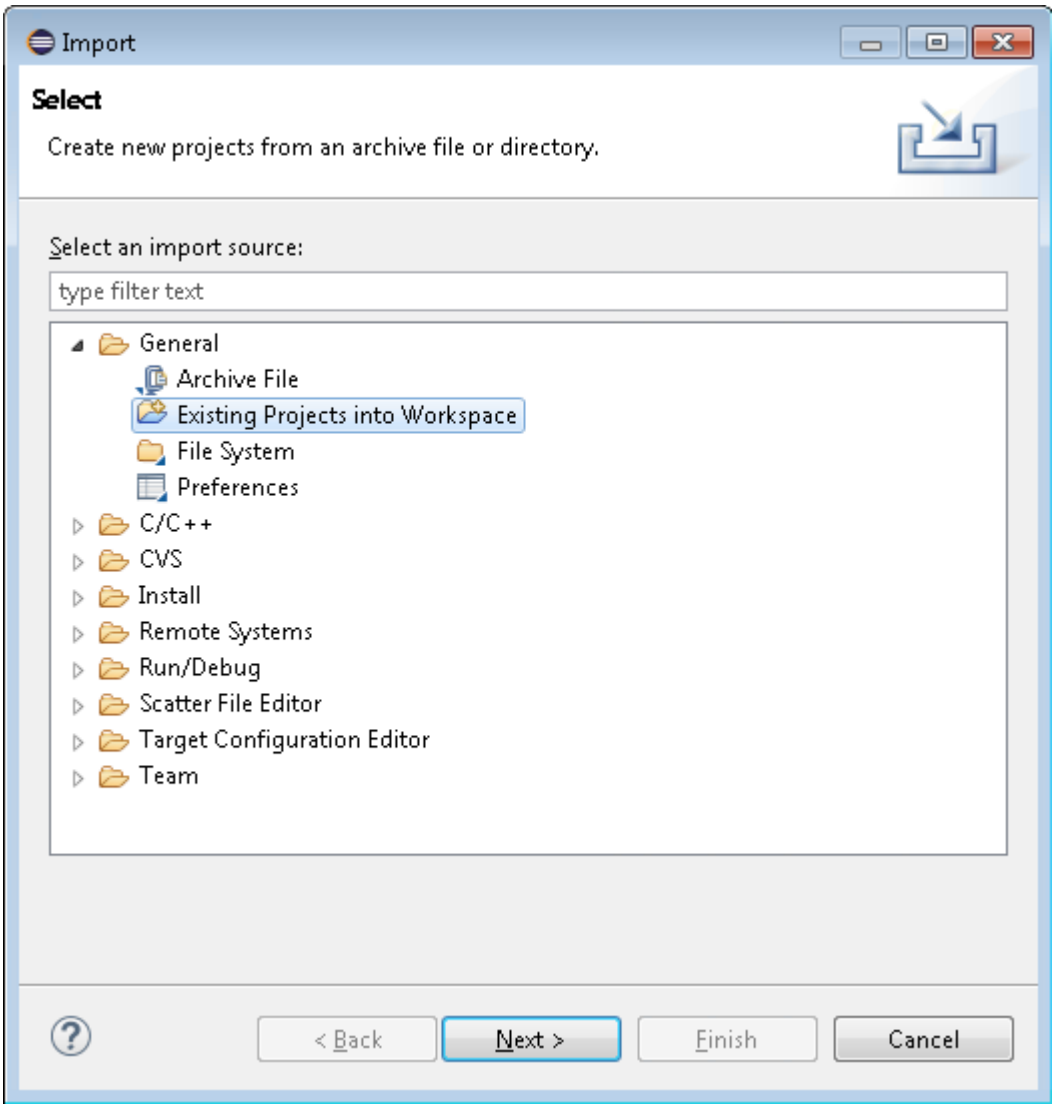
Before you begin

The Minimal Preloader (MPL) project provided in the Altera SoC EDS is a non-GPL Preloader that uses the Altera SoC EDS HWLIBs . To build the MPL, either the ARM Compiler Toolchain or the GCC must be greater than v14.1.

Import the MPL project by using the steps in the "Importing an Existing Bare Metal Project into DS-5" chapter.

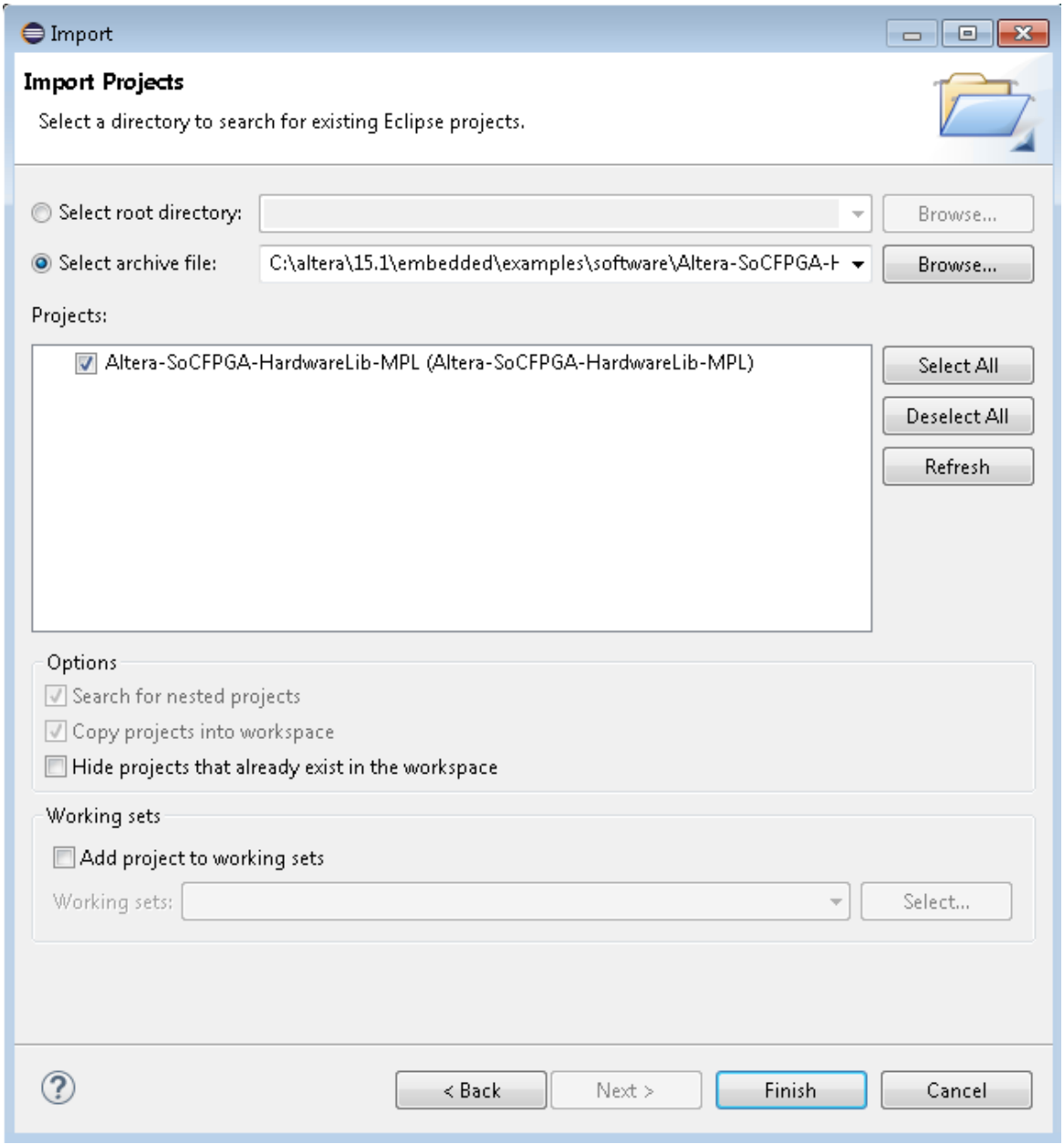
1. Go to **File** > **Import** and select **General** > **Existing Projects into Workspace**.

Figure 67: Import Existing Projects



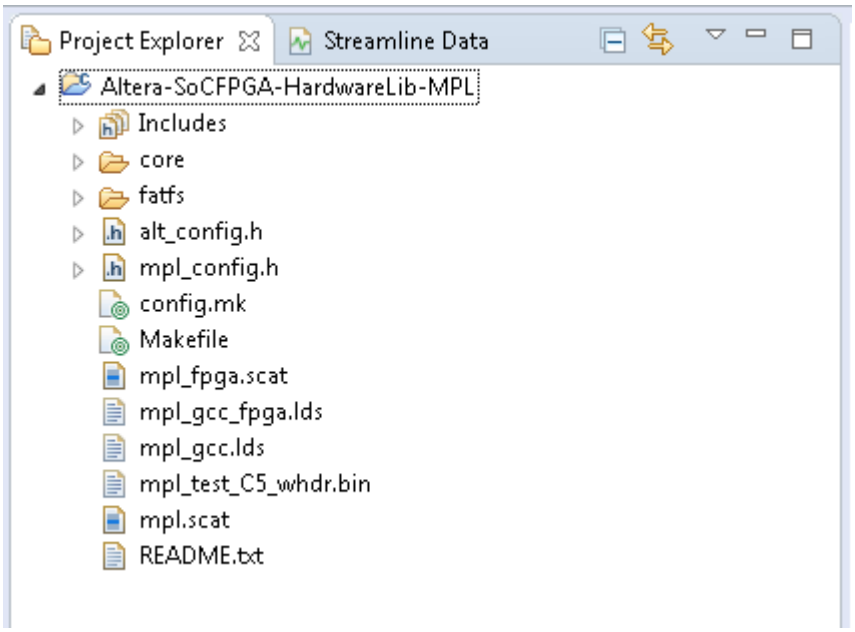
2. Select the **Altera-SoCFPGA-HardwareLib-MPL.tar** project from the <*SoC EDS installation path*>\`embedded\examples\software` and then click **Finish** to import the project.

Figure 68: Import MPL Project File



3. From the **Project Explorer** tab, verify all project files are present.

Figure 69: MPL Project File List



For more information about building and debugging, refer to the "Importing, Building and Debugging in a Make-Based Example" section.

Related Information

Importing, Building and Debugging in a Make-Based Example on page 40

Appendix: Troubleshooting

Debug Cable Does Not Work

Be sure that the USB-Blaster II driver is installed and that it is functional. Generally, following the development kit installation instructions and going through a few of the recommended examples is good enough to solve this issue.

FPGA Is Not Programmed Successfully

If this occurs, refer to specifics on your development kit, but usually this is due to a mismatched MSEL (programming mode).

Temporary Directories not Writable

Though uncommon, it is possible for temporary directories that the Eclipse-based DS-5 debugger relies on being writable are sometimes un-writable. Also, if you run into strange “permission denied” issues when starting debug sessions, search through all temporary directories in your environment **(tmp, TMP, temp, TEMP)** and change or set them to a writable directory.