XAPP1185 (v1.0) November 18, 2013

Zynq-7000 Platform Software Development Using the ARM® DS-5 Toolchain

Author: Simon George and Prushothaman Palanichamy

Summary

This document provides guidance on using the ARM Development Studio 5 (DS-5) design suite for the development, build, and debug of bare metal software for the Xilinx Zynq-7000 All Programmable SoC, which is based on the ARM Cortex™-A9 processor. This process is described in the following steps:

1. Xilinx SDK: Board Support Package (BSP) creation for custom hardware design
2. DS-5 tools: BSP import and build
3. DS-5 tools: Application build
4. Xilinx SDK and DS-5 tools: Zynq device FSBL Changes for ARMCC build
5. DS-5 tools: Debugger target configuration for Zynq device custom design

Introduction

This document assumes that you have a basic understanding of the architecture, boot flow, and associated Xilinx design tools for the Zynq-7000 product family as well as a basic understanding of the ARM DS-5 toolchain.It also assumes that the development host operating system is Windows. To complete the procedure in this application note, you must have installed the following host-based software tools:

- Xilinx Software Development Kit (SDK) 2013.3, which you can download at this link.
- ARM Development Studio 5 (DS-5) tool suite, which you can download at this link.

  Note: A 30-day evaluation is available for the DS-5 tool suite.

Currently Xilinx only supports the ARM commercial C and C++ compiler, not the Linaro GCC that is also distributed with the DS-5 tool suite.

Associated Design Files

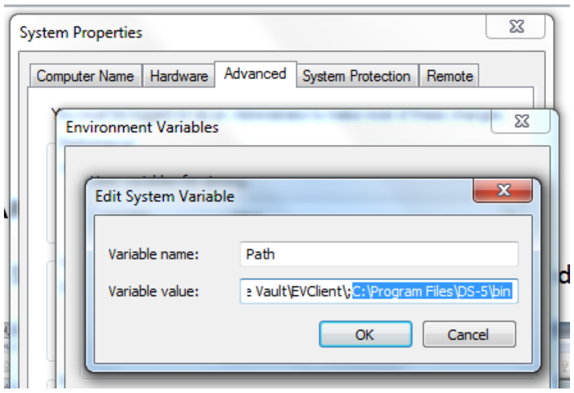
You need to customize the 2013.3 Standalone Board Support Package and First Stage Boot Loader (FSBL) template distributed by Xilinx. The design files for this application note (available at this link) provide these customizations as well as updated repositories. The design files also include a target initialization file and linker script example for use in the DS-5 tools, as described by this document. The Standalone BSP included in the design files is similar to the standalone_v3_11_a available from 2013.3 with a few changes to inline assembly functions.

Download and unzip the design files. Note the directory name and location where you save the files.

Tools Configuration

The procedure in this application note requires that you use the ARM compiler toolchain from within the Xilinx SDK environment, which necessitates that the corresponding ARM executables need to be visible outside of the DS-5 tool environment and its command prompt. To achieve this visibility, you need to modify the Windows system environment variables as shown in Figure 1.

Ensure that the appended path (shown highlighted in the illustration) matches your actual install location.



Variable name: Path

Variable value: e Vault\EVClient\;C:\Program Files\DS-5\bin

Figure 1: **Edit Windows System Environment Variable**

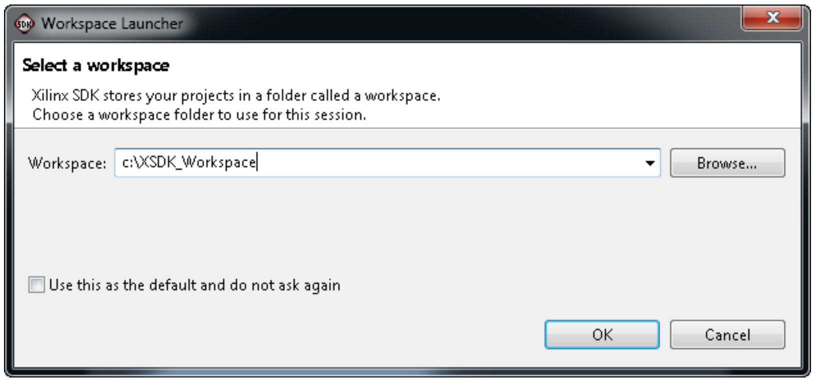
Step 1 - Xilinx SDK: Standalone Board Support Package Creation

Xilinx SDK dynamically assembles a customized BSP based on the selected hardware design, whether that is a customized design imported from Vivado® design suite or a preconfigured platform. This assembled BSP includes a collection of CPU complex-specific configuration and startup code along with platform address and configuration information for both hardened processing system and soft, programmable logic peripherals, which is effectively everything needed to link against when creating an application project.

You complete the physical process of creating this data-driven BSP through a Library Generator (libgen) utility, which is an integral part of the Xilinx SDK environment. Therefore, development in the ARM DS-5 tools must also baseline from this BSP. However, there is GCC-specific assembler code included in the default start-up files, which therefore require customization. Part of this process is automated. The design files included with this application note patch the files for this release.

Use the following steps to build the patched Standalone BSP repository from Xilinx SDK.

1. Invoke Xilinx SDK, then create and configure the workspace.
    a. Open Xilinx SDK and create a new workspace (suggested name: XSDK_Workspace).



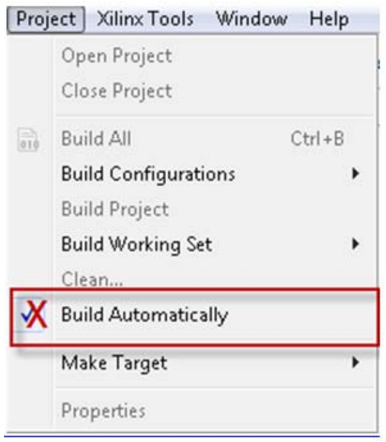
Select a workspace

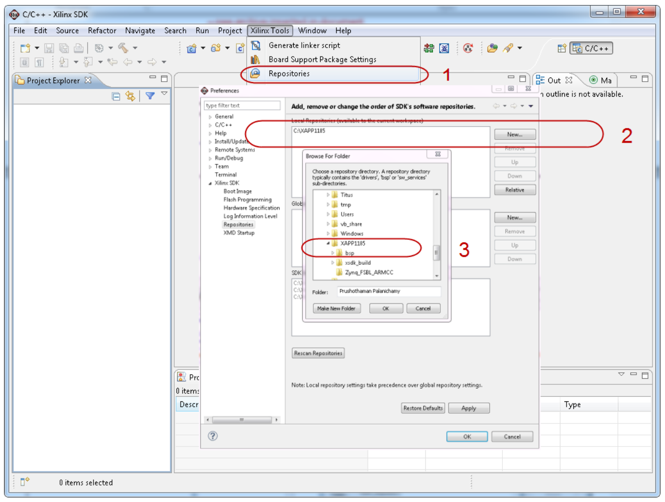
Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: c:\XSDK_Workspace

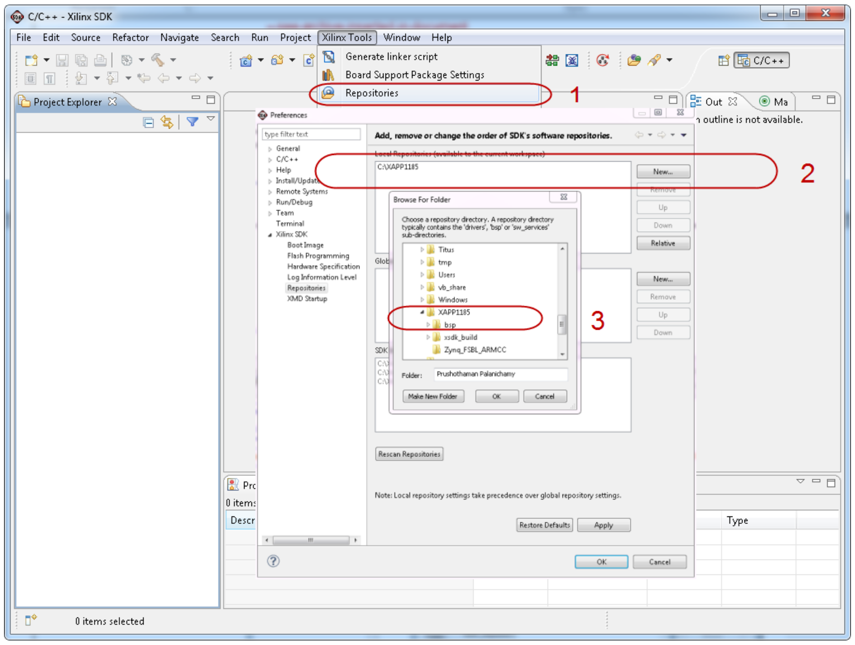
☐ Use this as the default and do not ask again

b. In Xilinx SDK, click **Project** and disable the **Build Automatically** option. This setting allows you to change the compiler option before building a project.





c. Click **Xilinx Tools** > **Repositories** and click **New** on the **Local Repositories** box. Browse to the `XAPP1185` folder in the application note design files and add the patched Standalone BSP as new repository.

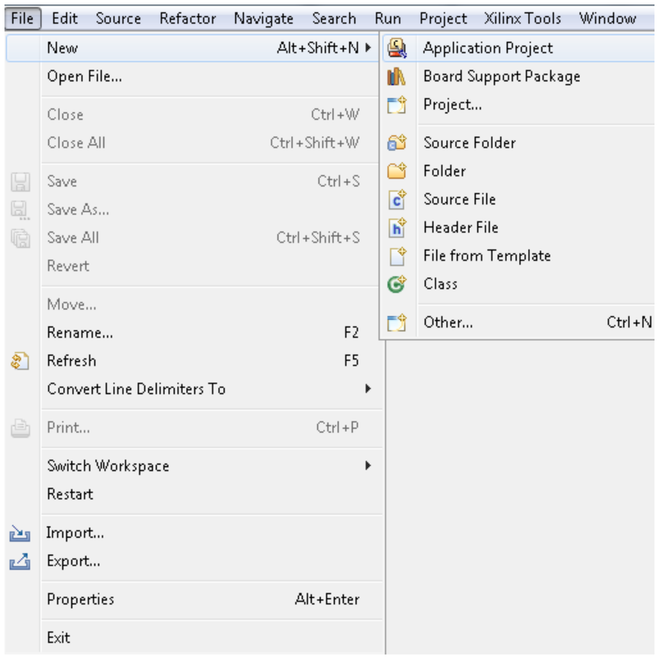

C/C++ - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
Generate linker script
Board Support Package Settings
Repositories
1
Preferences
Add, remove or change the order of SDK's software repositories.
C:\XAPP1185
New...
2
Browse For Folder
Choose a repository directory. A repository directory typically contains the 'drivers', 'bsp' or 'sw_services' sub-directories.
XAPP1185
bsp
xsdk_build
Zynq_FSBL_ARMCC
3
Folder:
Make New Folder
OK
Cancel
Rescan Repositories
Note: Local repository settings take precedence over global repository settings.
Restore Defaults
Apply
0 items selected

2. Create a new Standalone application project and Standalone BSP (ARMCC version).

   XSDK provides a New Project creation wizard, which guides you through the process of creating a new project in a coordinated way. This wizard is used in the following illustrations. Project settings may deviate from what is shown based on individual needs.

   This application note uses the Hello World application example created in Xilinx SDK as the first ‘C’ application to run on the target built through the DS-5 tools. If you do not need to create a sample application project, you can either individually create a BSP against a previously imported hardware platform or manually create a hardware platform and then create a BSP against it.

a. In Xilinx SDK, click **File** > **New** > **Application Project**.



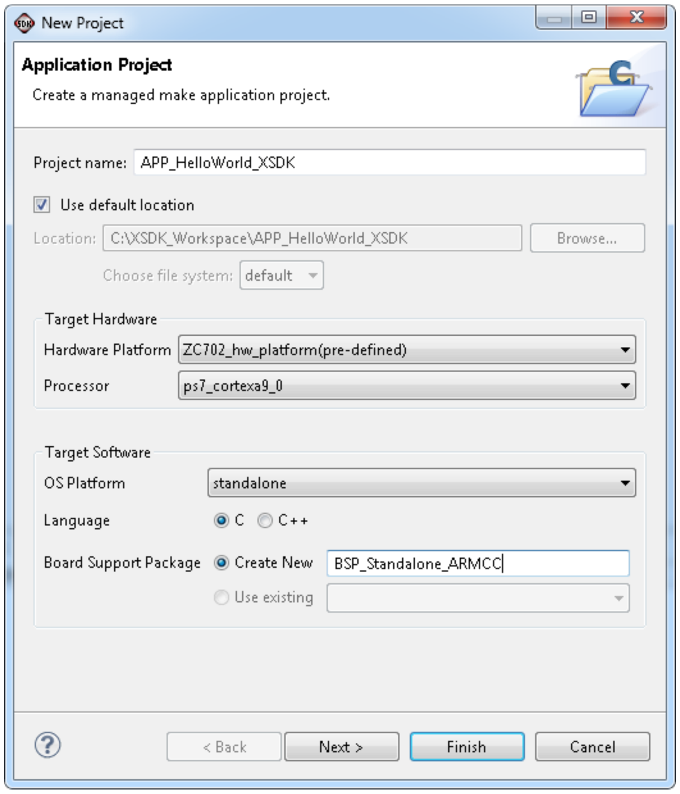
b. In the Application Project screen make the following settings.

- Hardware Platform

  Select either predefined or custom (Create New) in the pull-down menu. If you select **Create New**, SDK prompts you to navigate to the location of the `<HardwareDesign.xml>` provided by your hardware development team, a directory that also includes `ps7_init .c`/`.h` and `.tcl`.

- Software Platform

  Target Software: Board Support Package. The suggested name for the Board Support Package is `BSP_Standalone_ARMCC`, as shown in the following illustration . (Xilinx SDK uses the Xilinx-distributed GCC to build the BSP source by default, but you modify the toolchain options later in this procedure.)



Project name: APP_HelloWorld_XSDK

[✓] Use default location

Location: C:\XSDK_Workspace\APP_HelloWorld_XSDK

Choose file system: default

Target Hardware

Hardware Platform: ZC702_hw_platform(pre-defined)

Processor: ps7_cortexa9_0

Target Software

OS Platform: standalone

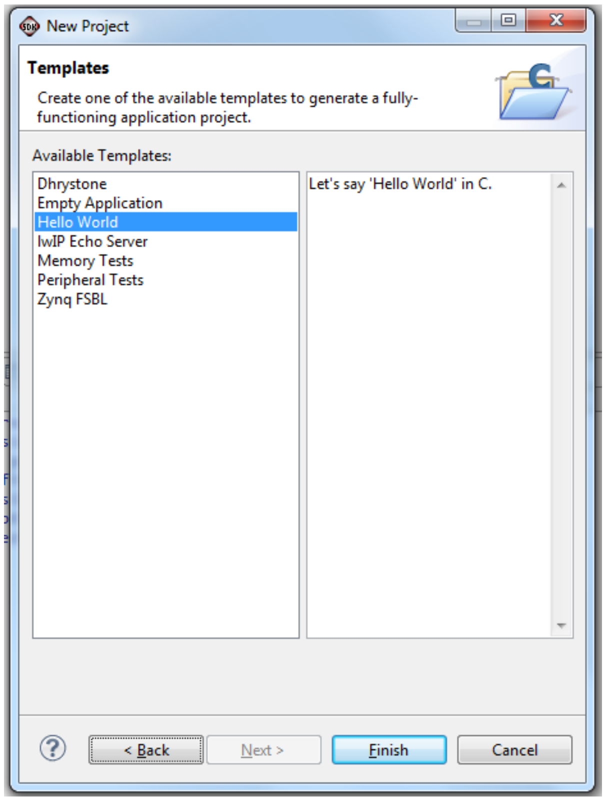
Language: [✓ C] C++

Board Support Package: [✓ Create New] BSP_Standalone_ARMCC

- Click **Next**.
- Set the application template.

  You can use any of the application templates to validate the build flow before undertaking your application. This application note uses the “Hello World” application.

- Click **Finish**.



New Project

Templates

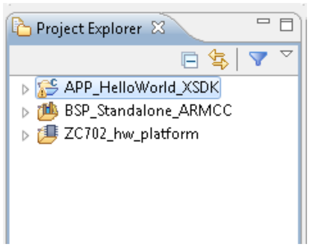
Create one of the available templates to generate a fully-functioning application project.

Available Templates:

Dhrystone
Empty Application
Hello World
lwIP Echo Server
Memory Tests
Peripheral Tests
Zynq FSBL

Let's say 'Hello World' in C.

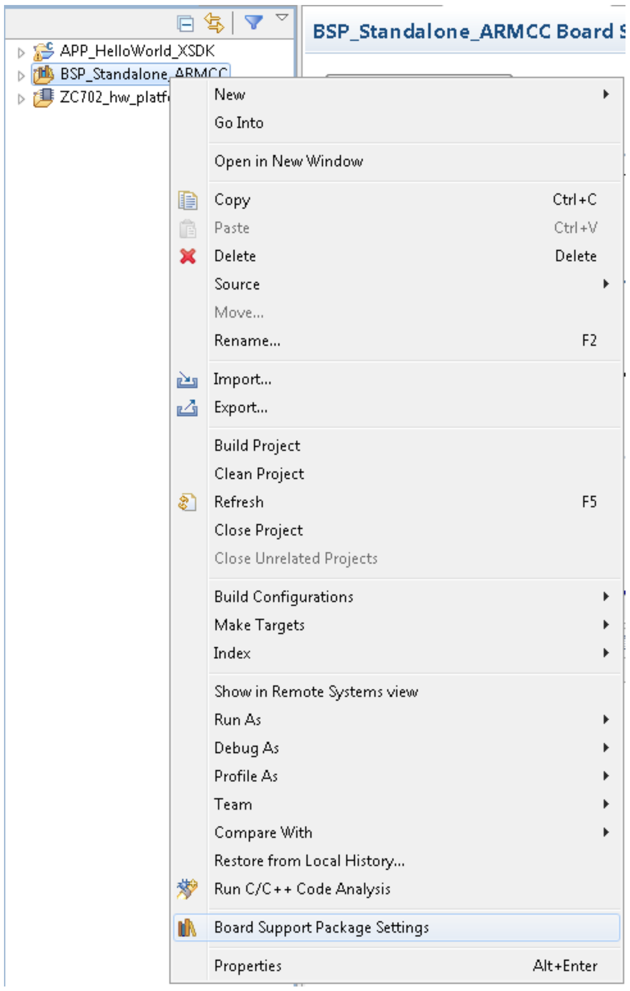
The tools create the files shown in the following illustration:



Project Explorer

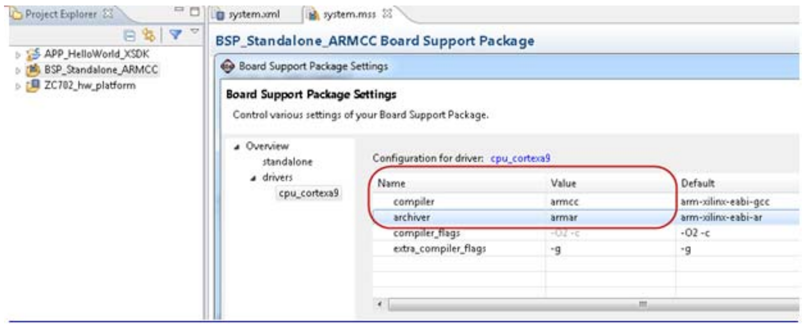
APP_HelloWorld_XSDK
BSP_Standalone_ARMCC
ZC702_hw_platform

c. Right-click the BSP project, **BSP_Standalone_ARMCC**, and then select **Board Support Package Settings**.



d. Navigate to **drivers** > **cpu_cortexa9** and change the following parameters.

- compiler: armcc
- archiver: armar



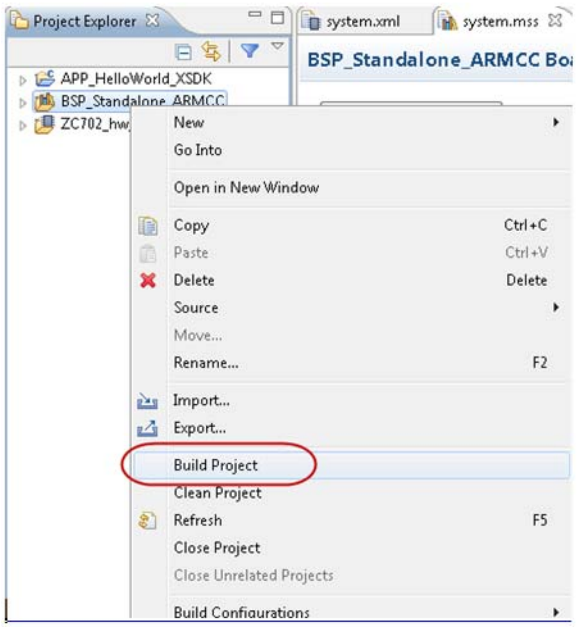
Board Support Package Settings

Control various settings of your Board Support Package.

Configuration for driver: cpu_cortexa9

Name	Value	Default
compiler	armcc	arm-xilinx-eabi-gcc
archiver	armar	arm-xilinx-eabi-ar
compiler_flags	-O2 -c	-O2 -c
extra_compiler_flags	-g	-g

e. Right-click the BSP project and select **Build Project**.



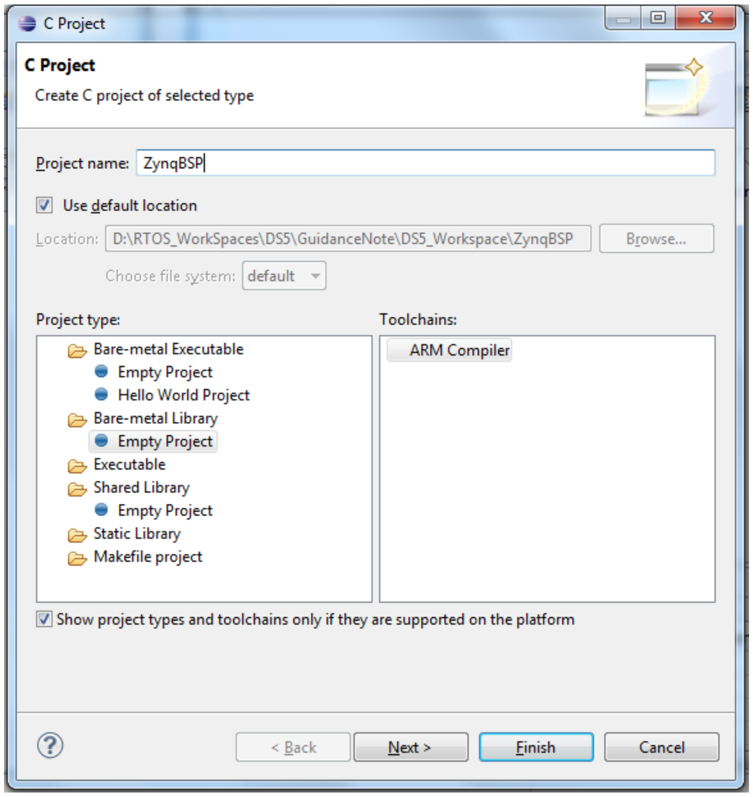
f. The patched BSP source repository is now built against the ZC702_hw_platform (pre-defined using the ARMCC toolchains).

Note: You don't need to set additional compiler options. You could use the archived library. However, in the next steps you will import the customized BSP source tree into the DS-5 tools where many options are exposed as build options.

Step 2 - DS-5 Tools: BSP Import and Build

1. Invoke **Eclipse for DS-5** to create a new workspace. The example in this document creates a new workspace in `C:\DS5_Workspace`.

2. Import the Standalone BSP
   a. In the DS-5 tool, click **File** > **New** > **C Project** > **Bare metal Library** > **Empty Project**.


C Project

C Project

Create C project of selected type

Project name: ZynqBSP

☑ Use default location

Location: D:\RTOS_WorkSpaces\DS5\GuidanceNote\DS5_Workspace\ZynqBSP

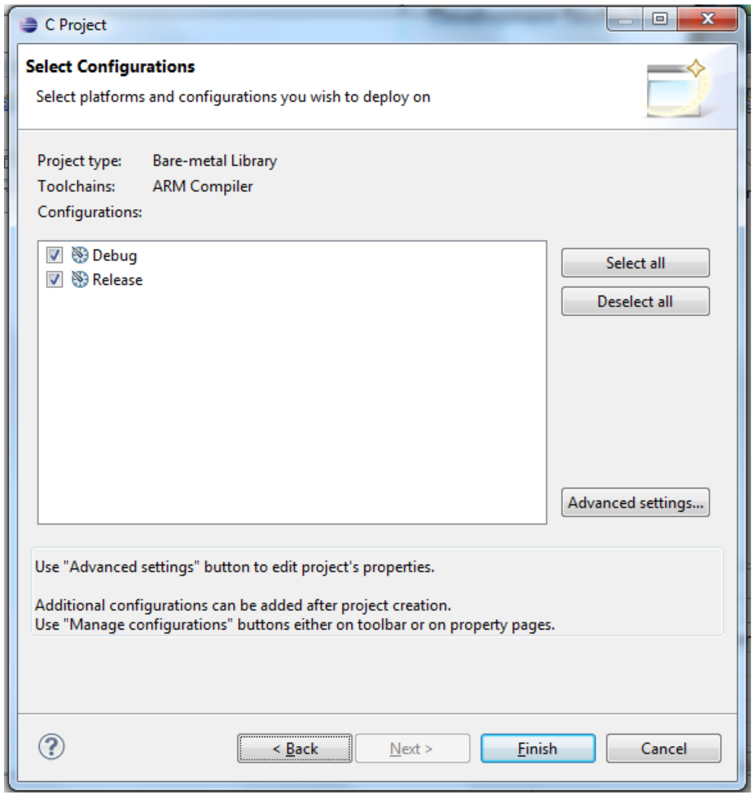
Choose file system: default

Project type:
- Bare-metal Executable
  - Empty Project
  - Hello World Project
- Bare-metal Library
  - Empty Project
- Executable
- Shared Library
  - Empty Project
- Static Library
- Makefile project

Toolchains:
- ARM Compiler

☑ Show project types and toolchains only if they are supported on the platform

b. Name the project (suggested name: “ZynqBSP”). Click **Next**.



C Project

Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Bare-metal Library
Toolchains: ARM Compiler
Configurations:

- [✓] Debug
- [✓] Release

Select all | Deselect all | Advanced settings...

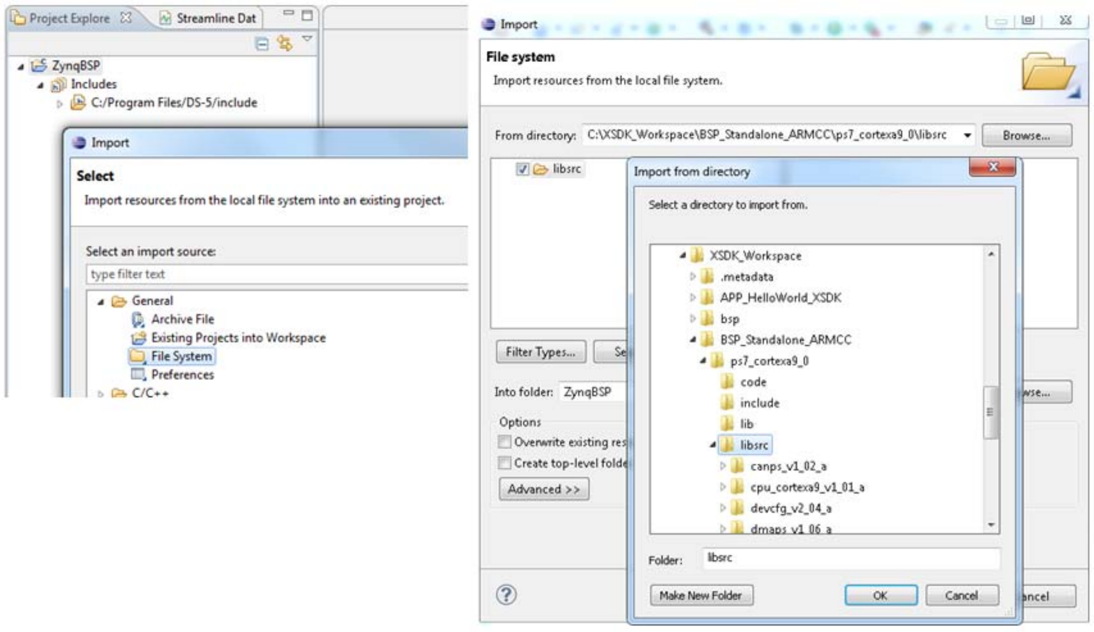
Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

< Back | Next > | Finish | Cancel

c. Click **Finish**.

d. Right-click the Library project and import the Xilinx-SDK-generated (ARMCC) BSP source tree, specifically the directory tree below “libsrc.” Browse to the folder `BSP_Standalone_ARMCC\ps7_cortexa9_0\libsrc`, then click **OK**.



Project Explore | Streamline Dat

- ZynqBSP
  - Includes
    - C:/Program Files/DS-5/include

Import

Select

Import resources from the local file system into an existing project.

Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- C/C++

Import

File system

Import resources from the local file system.

From directory: C:\XSDK_Workspace\BSP_Standalone_ARMCC\ps7_cortexa9_0\libsrc | Browse...

[✓] libsrc

Filter Types... | Se

Into folder: ZynqBSP

Options

- [ ] Overwrite existing res
- [ ] Create top-level folde

Advanced >>

Import from directory

Select a directory to import from.

- XSDK_Workspace
  - .metadata
  - APP_HelloWorld_XSDK
  - bsp
  - BSP_Standalone_ARMCC
    - ps7_cortexa9_0
      - code
      - include
      - lib
      - libsrc
        - canps_v1_02_a
        - cpu_cortexa9_v1_01_a
        - devcfg_v2_04_a
        - dmaps_v1_06_a

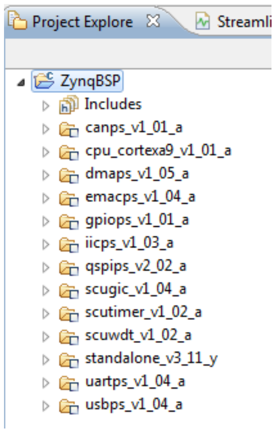
Folder: libsrc

Make New Folder | OK | Cancel

e. In the Import window, Click **Finish**.

Note: By using links to the original sources, reference file updates will propagate into the DS-5 tools.

f. The folders shown in the following illustration are now included in the ZynqBSP project.

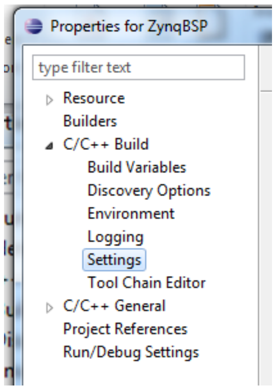


ZynqBSP
- Includes
- canps_v1_01_a
- cpu_cortexa9_v1_01_a
- dmaps_v1_05_a
- emacps_v1_04_a
- gpiops_v1_01_a
- iicps_v1_03_a
- qspips_v2_02_a
- scugic_v1_04_a
- scutimer_v1_02_a
- scuwdt_v1_02_a
- standalone_v3_11_y
- uartps_v1_04_a
- usbps_v1_04_a

3. Change the build settings to suit the Zynq architecture.

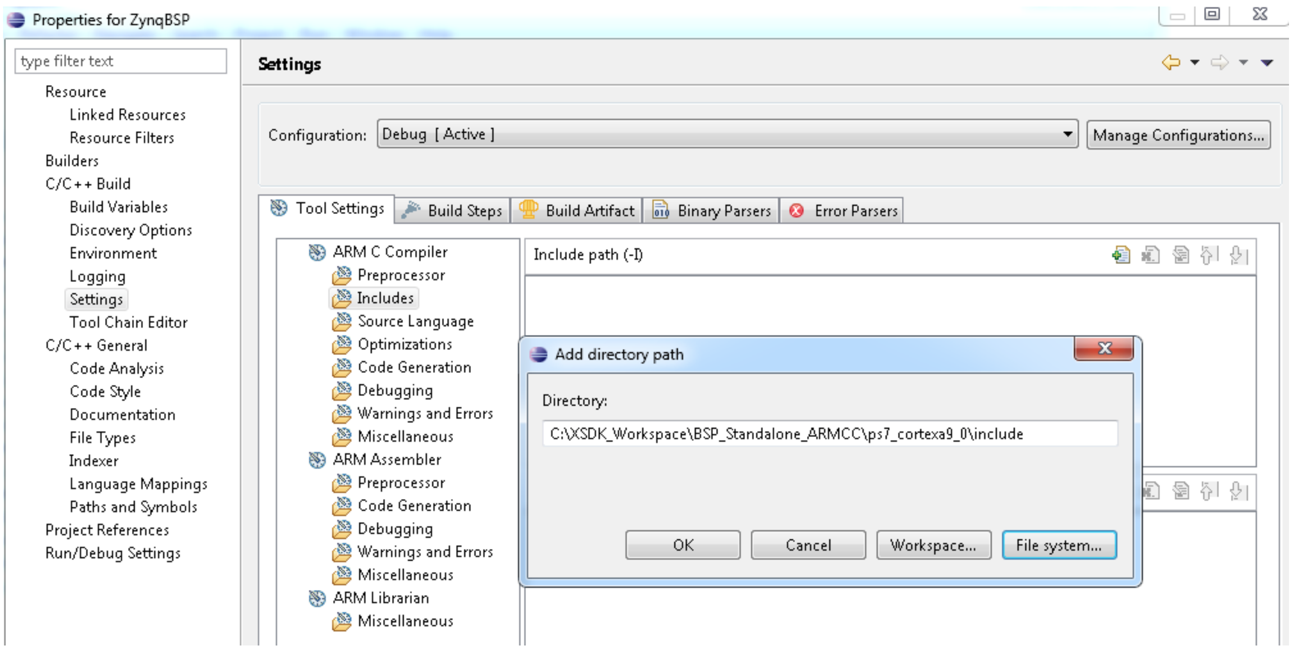
a. Right-click on the project (**ZynqBSP**), then click **Properties**.

b. In the Properties wizard, navigate to **C/C++ Build** > **Settings**, then modify the following settings of the ARM Compiler and ARM Assembler.



- In the ARM C Compiler, set the include paths for the referenced header files generated by the Xilinx SDK libgen process, customized to the hardware configuration:

```
-I <XSDK_workspace>\BSP_Standalone_ARMCC\ps7_cortexa9_[0]\include
```



Properties for ZynqBSP — Settings

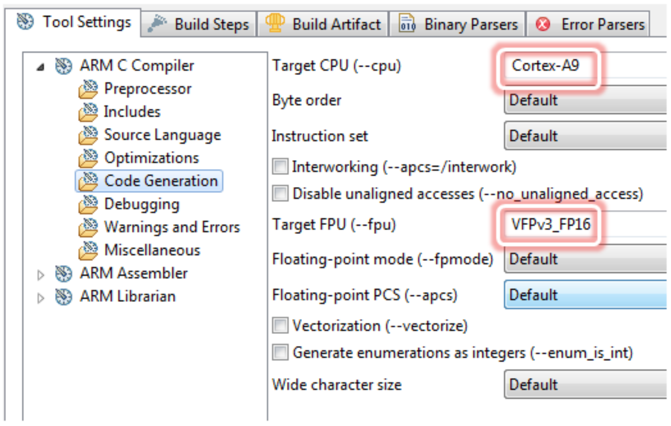
Configuration: Debug [Active]

Tool Settings > ARM C Compiler > Includes — Include path (-I)

Add directory path

Directory: C:\XSDK_Workspace\BSP_Standalone_ARMCC\ps7_cortexa9_0\include

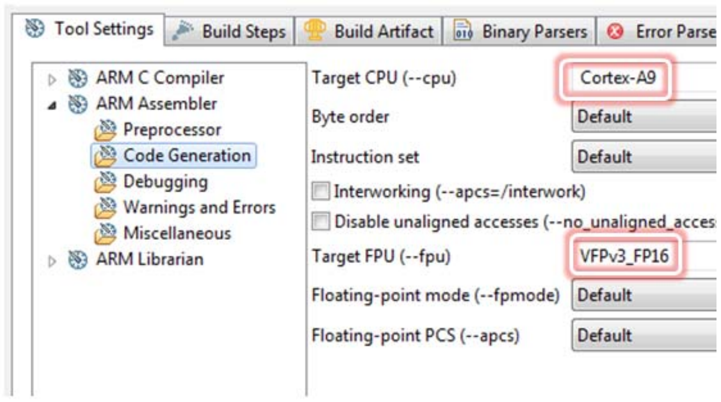
- In the ARM C Compiler, set the Code Generation parameters specific to the Zynq device:
  - --cpu Cortex-A9
  - --fpu VFPv3_FP16



Tool Settings > ARM C Compiler > Code Generation

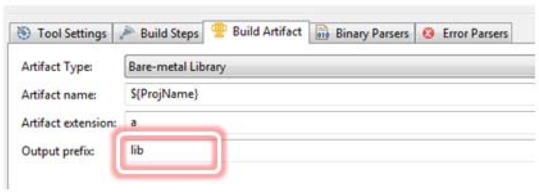
Target CPU (--cpu)	Cortex-A9
Byte order	Default
Instruction set	Default
☐ Interworking (--apcs=/interwork)	
☐ Disable unaligned accesses (--no_unaligned_access)	
Target FPU (--fpu)	VFPv3_FP16
Floating-point mode (--fpmode)	Default
Floating-point PCS (--apcs)	Default
☐ Vectorization (--vectorize)	
☐ Generate enumerations as integers (--enum_is_int)	
Wide character size	Default

- In the ARM Assembler, set the Code Generation parameters specific to the Zynq device:
  - --cpu Cortex-A9
  - --fpu VFPv3_FP16



Target CPU (--cpu) Cortex-A9
Byte order Default
Instruction set Default
Interworking (--apcs=/interwork)
Disable unaligned accesses (--no_unaligned_acces
Target FPU (--fpu) VFPv3_FP16
Floating-point mode (--fpmode) Default
Floating-point PCS (--apcs) Default

c. Click the **Build Artifacts** tab and make the following settings:
  - Artifact extension: a
  - Output Prefix: lib



Artifact Type: Bare-metal Library
Artifact name: ${ProjName}
Artifact extension: a
Output prefix: lib

d. Click the Build button. If the build is successful, the tool reports:

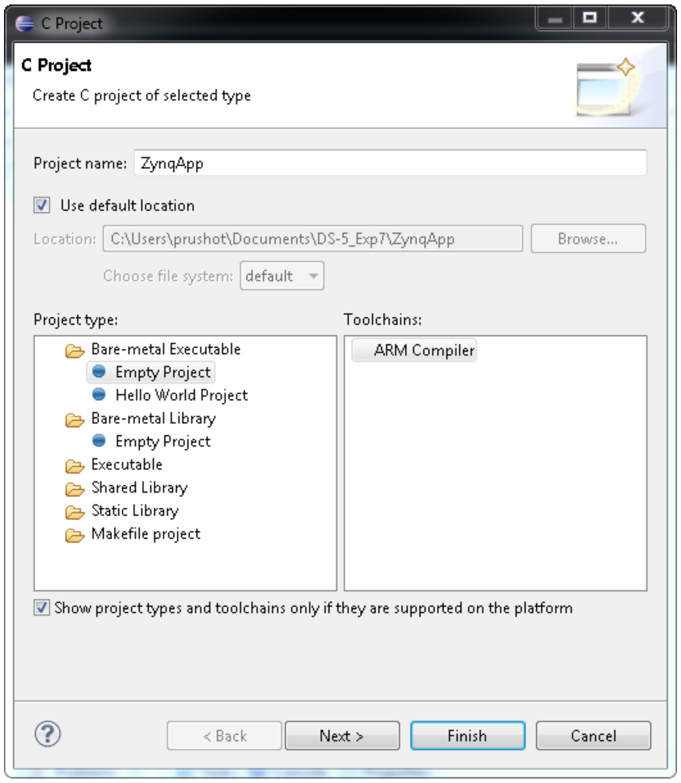
```
'Finished building target: libZynqBSP.a'
' '

**** Build Finished ****
```

You have now completed this step.

Step 3 - DS-5 Tools: Application Build

Use this procedure to create a DS-5 application project for the Zynq device.

1. Import and create a software application:
   a. In the DS-5 tool, click **File** > **New** > **C Project** > **Bare metal Executable** > **Empty Project**.



C Project

Create C project of selected type

Project name: ZynqApp

☑ Use default location

Location: C:\Users\prushot\Documents\DS-5_Exp7\ZynqApp

Choose file system: default

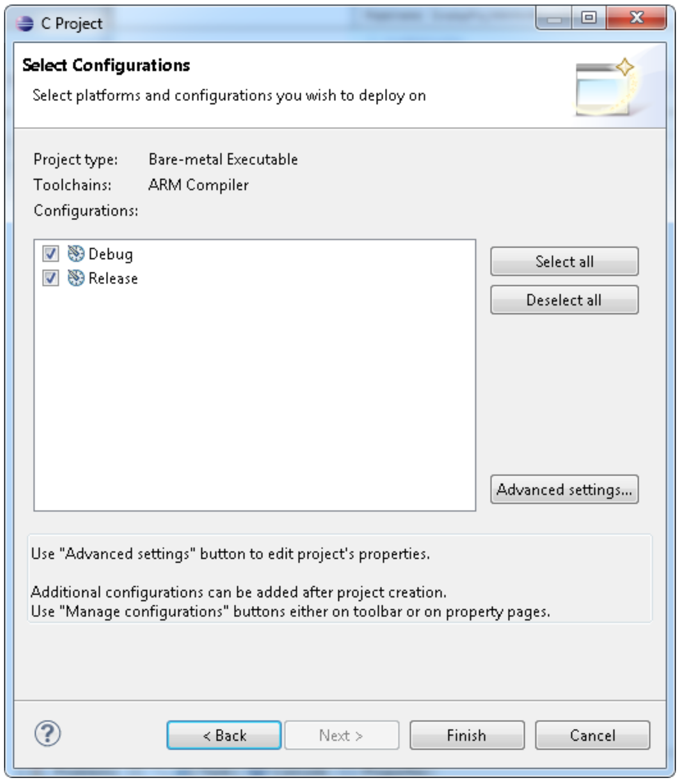
Project type:
- Bare-metal Executable
  - Empty Project
  - Hello World Project
- Bare-metal Library
  - Empty Project
- Executable
- Shared Library
- Static Library
- Makefile project

Toolchains:
- ARM Compiler

☑ Show project types and toolchains only if they are supported on the platform

   b. Name the Project (suggested name: “ZynqAPP”).

c. Click **Next**.



C Project

Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Bare-metal Executable
Toolchains: ARM Compiler
Configurations:

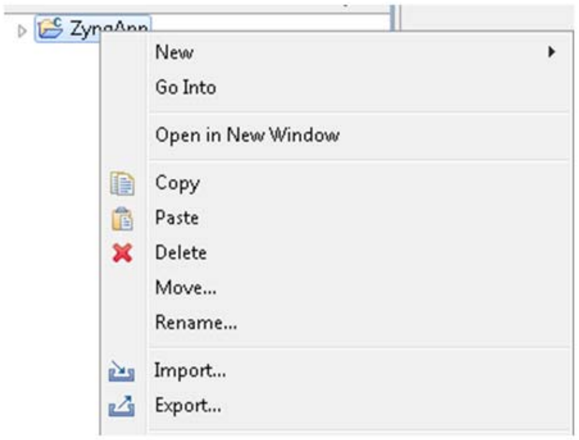
- [✓] Debug
- [✓] Release

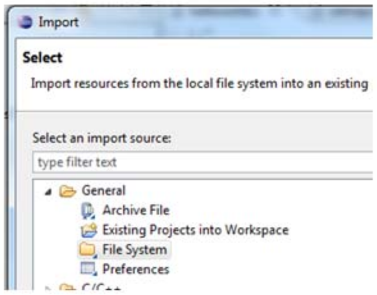
Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

d. Click **Finish** to close the C Project wizard.

e. Right-click the application project, select **Import**, and navigate to the “Hello World” application project (App_HelloWorld_XSDK) created in Step 1 of this application note.





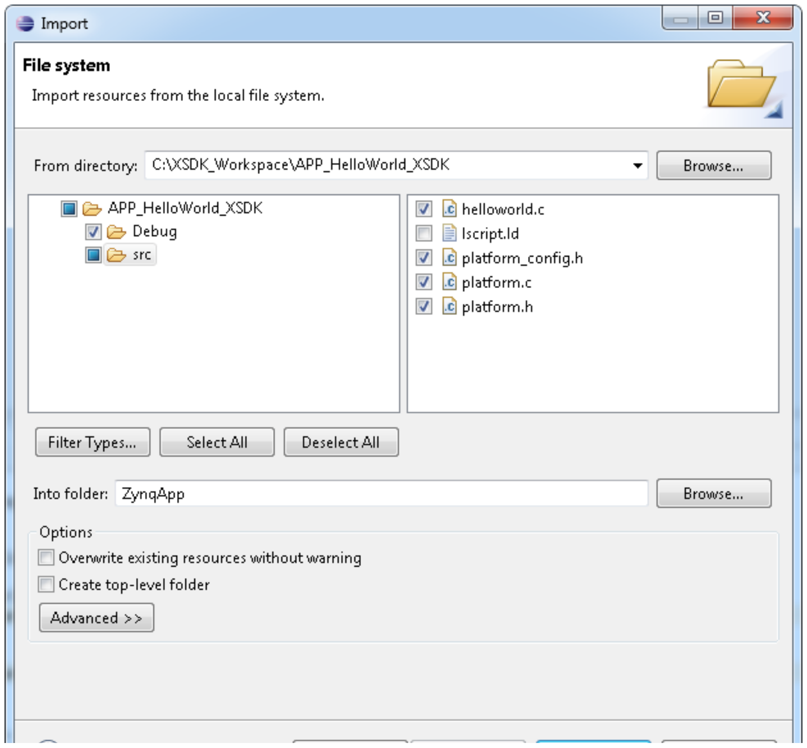
Import

Select

Import resources from the local file system into an existing

Select an import source:

f. Include all the C Source and Header files (*.c, *.h, *.S).


Import

File system

Import resources from the local file system.

From directory: C:\XSDK_Workspace\APP_HelloWorld_XSDK

APP_HelloWorld_XSDK
- ✓ Debug
- src

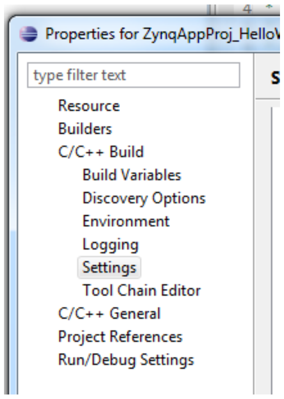
- ✓ helloworld.c
- lscript.ld
- ✓ platform_config.h
- ✓ platform.c
- ✓ platform.h

Into folder: ZynqApp

Options
- Overwrite existing resources without warning
- Create top-level folder

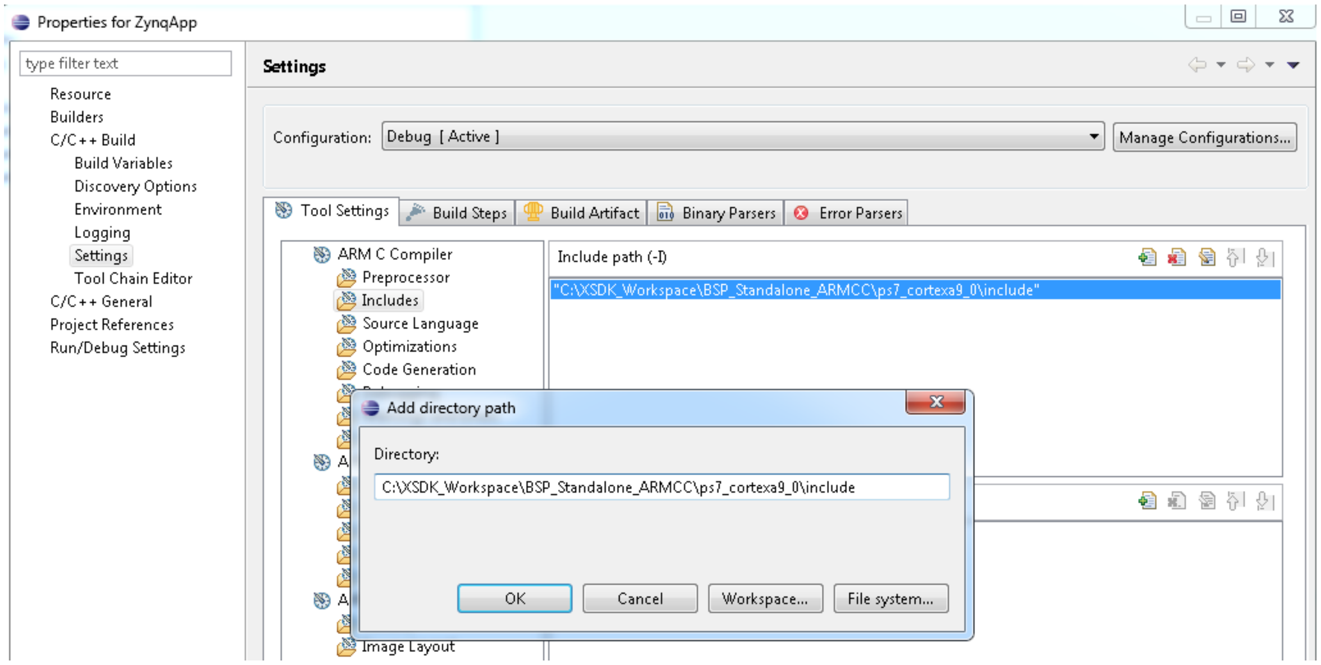
2. Change the build settings to suit the Zynq architecture.

   a. Right-click the project (ZynqApp) and select **Properties**. In the Properties wizard, navigate to **C/C++ Build** > **Settings** and modify the following settings of the ARM Compiler, ARM Assembler, and ARM Linker:



- In the ARM C Compiler, set the Include paths for referenced header files generated by the Xilinx SDK libgen process, customized to the hardware configuration:

```
../../../XSDK_Workspace/BSP_Standalone_ARMCC/ps7_cortexa9_0/include
```



Configuration: Debug [Active]

Include path (-I)

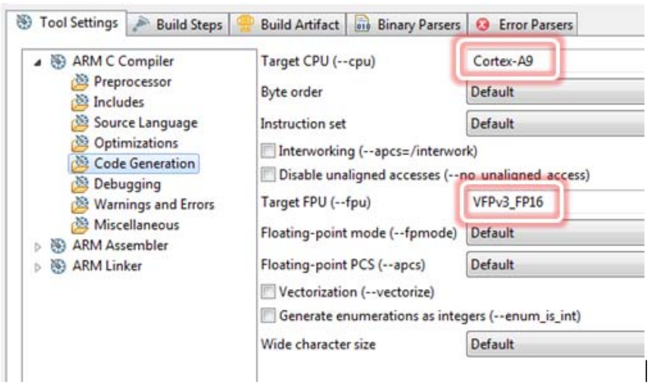
"C:\XSDK_Workspace\BSP_Standalone_ARMCC\ps7_cortexa9_0\include"

Directory:

C:\XSDK_Workspace\BSP_Standalone_ARMCC\ps7_cortexa9_0\include

b. In the ARM C Compiler, set the Code Generation parameters specific to Zynq architecture:

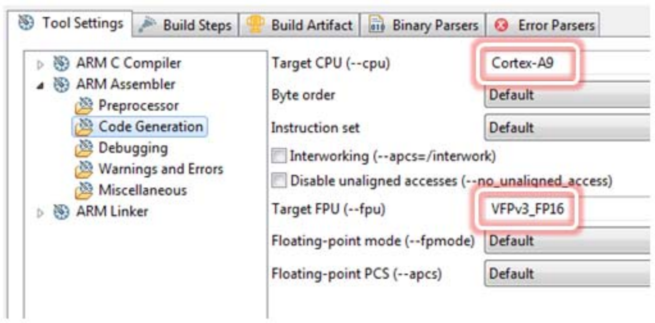
- --cpu Cortex-A9
- --fpu VFPv3_FP16



Target CPU (--cpu)	Cortex-A9
Byte order	Default
Instruction set	Default
☐ Interworking (--apcs=/interwork)	
☐ Disable unaligned accesses (--no_unaligned_access)	
Target FPU (--fpu)	VFPv3_FP16
Floating-point mode (--fpmode)	Default
Floating-point PCS (--apcs)	Default
☐ Vectorization (--vectorize)	
☐ Generate enumerations as integers (--enum_is_int)	
Wide character size	Default

c. In the ARM Assembler, set the Code Generation parameters specific to Zynq architecture:

- --cpu Cortex-A9
- --fpu VFPv3_FP16



Target CPU (--cpu) Cortex-A9
Byte order Default
Instruction set Default
Interworking (--apcs=/interwork)
Disable unaligned accesses (--no_unaligned_access)
Target FPU (--fpu) VFPv3_FP16
Floating-point mode (--fpmode) Default
Floating-point PCS (--apcs) Default

d. In the ARM Linker, set the General parameters specific to Zynq architecture:

- --entry vector_table
- --cpu Cortex-A9
- --fpu VFPv3_FP16

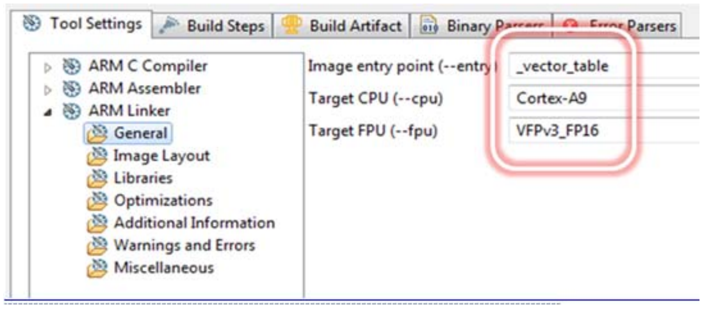
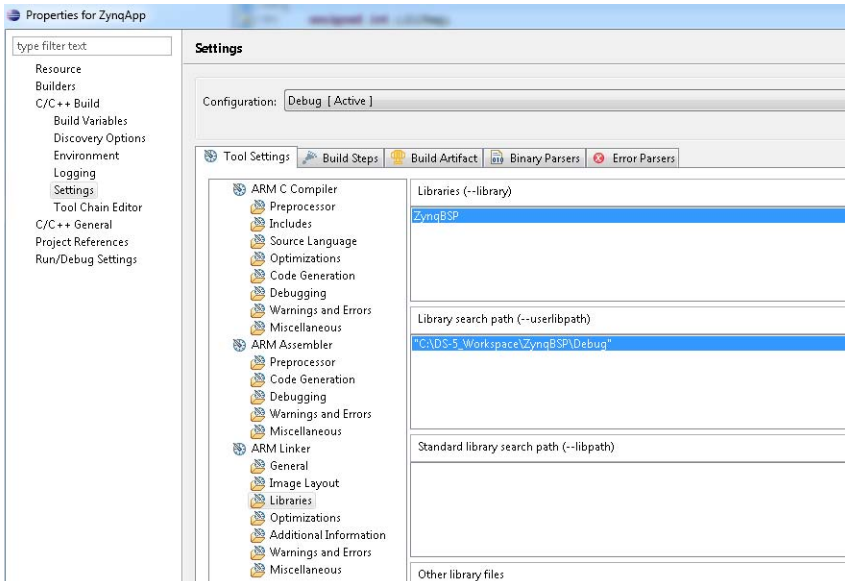


Image entry point (--entry) _vector_table
Target CPU (--cpu) Cortex-A9
Target FPU (--fpu) VFPv3_FP16

e. In the ARM Linker > Libraries, add the reference BSP build created in Step 2 of this application note:

- Libraries (--library): `ZynqBSP`
- Library Search Path (--userlibpath): `${workspace_loc:/ZynqBSP/Debug}`



Properties for ZynqApp

Settings

Configuration: Debug [Active]

Libraries (--library): ZynqBSP

Library search path (--userlibpath): "C:\DS-5_Workspace\ZynqBSP\Debug"

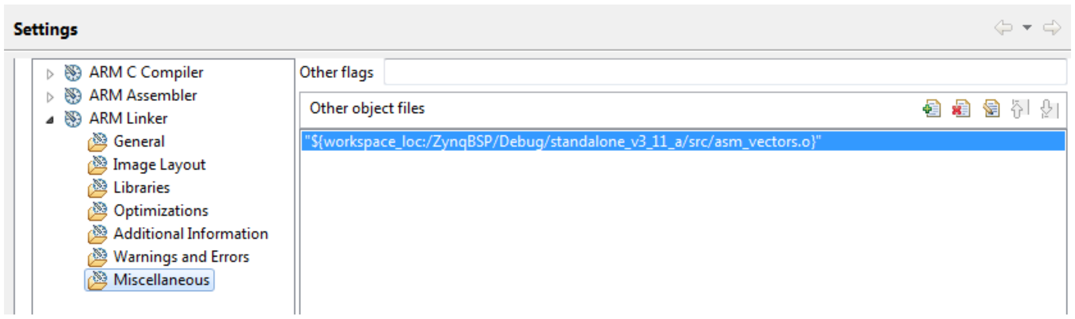
Standard library search path (--libpath)

Other library files

Note: The entry point matches the start of memory against which the program is linked in the scatter file. The definition of “_vector_table” can be found in `asm_vector.s`, which resides in the “standalone” code of the BSP source tree.

f. In the ARM Linker, select **Miscellaneous** > **Other objects files**.

g. In the Other object files tab, Click the **Add** button and navigate to `$workspace_loc:\ZynqBSP\Debug\ps7_cortexa9_0\libsrc\standalone_v3_11_a\src`” and select asm_vectors.o file

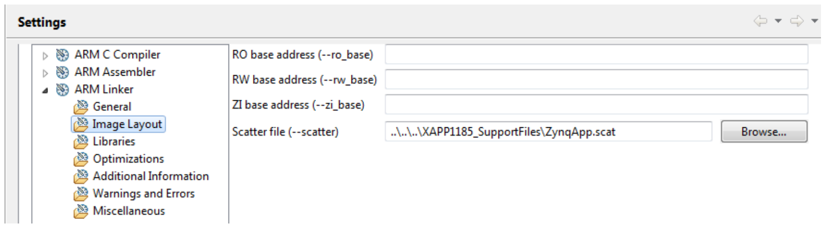


Settings

Other flags

Other object files: "${workspace_loc:/ZynqBSP/Debug/standalone_v3_11_a/src/asm_vectors.o}"

h. In the ARM Linker, select **Image Layout** and define the Linker Script, referred to here as a scatter file.



Settings

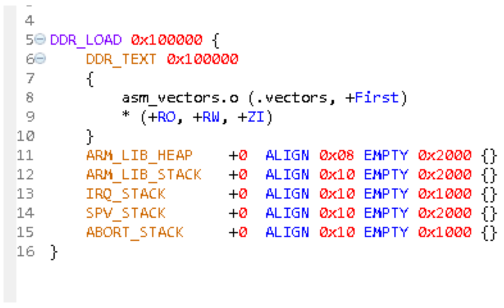
RO base address (--ro_base)

RW base address (--rw_base)

ZI base address (--zi_base)

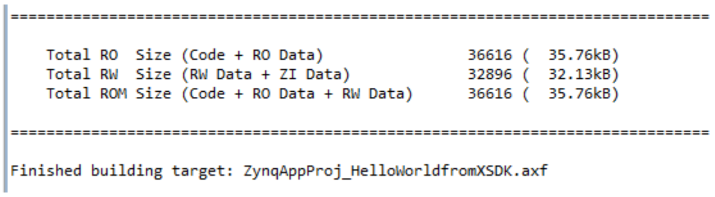
Scatter file (--scatter): ..\..\..\XAPP1185_SupportFiles\ZynqApp.scat

i. A Sample `ZynqApp.scat` file Image Layout is shown in the figure below. This file can be found in the XAPP1185 reference files.



```
DDR_LOAD 0x100000 {
    DDR_TEXT 0x100000
    {
        asm_vectors.o (.vectors, +First)
        * (+RO, +RW, +ZI)
    }
    ARM_LIB_HEAP    +0  ALIGN 0x08 EMPTY 0x2000 {}
    ARM_LIB_STACK   +0  ALIGN 0x10 EMPTY 0x2000 {}
    IRQ_STACK       +0  ALIGN 0x10 EMPTY 0x1000 {}
    SPV_STACK       +0  ALIGN 0x10 EMPTY 0x2000 {}
    ABORT_STACK     +0  ALIGN 0x10 EMPTY 0x1000 {}
}
```

j. Compile the application. As a reference, the following illustration shows the compiled size in the example flow.



```
==============================================================================

    Total RO  Size (Code + RO Data)                36616 (  35.76kB)
    Total RW  Size (RW Data + ZI Data)             32896 (  32.13kB)
    Total ROM Size (Code + RO Data + RW Data)      36616 (  35.76kB)

==============================================================================

Finished building target: ZynqAppProj_HelloWorldfromXSDK.axf
```

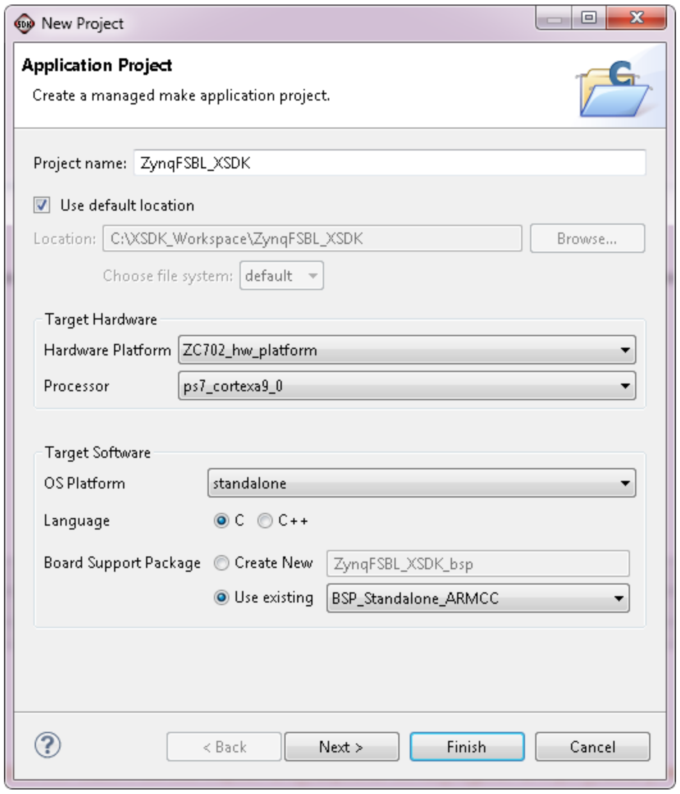
You have now completed this step.

Step 4 - Xilinx SDK and DS-5 Tools: Zynq Device FSBL Changes for ARMCC Build

Use the following steps to build the First Stage Boot Loader (FSBL) with ARMCC.

1. Open Xilinx SDK, and use the existing workspace, “XSDK_Workspace.”
2. Click **File** > **New Application Project**.

3. Name the project (suggested name: “ZynqFSBL_XSDK”). Use the existing BSP (BSP_Standalone_ARMCC).



Application Project

Create a managed make application project.

Project name: ZynqFSBL_XSDK

[✓] Use default location

Location: C:\XSDK_Workspace\ZynqFSBL_XSDK

Choose file system: default

Target Hardware

Hardware Platform: ZC702_hw_platform

Processor: ps7_cortexa9_0

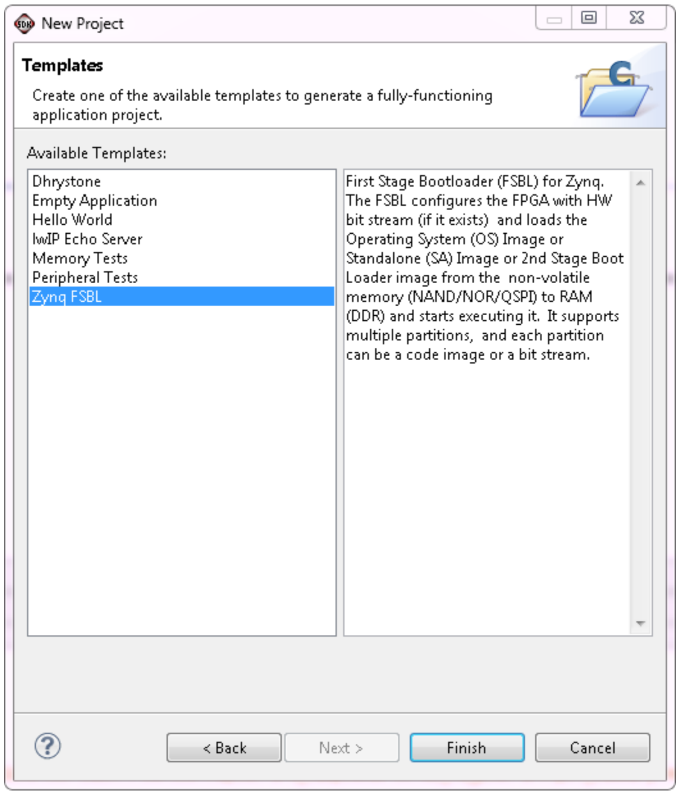
Target Software

OS Platform: standalone

Language: [✓ C] C++

Board Support Package: Create New ZynqFSBL_XSDK_bsp [✓ Use existing] BSP_Standalone_ARMCC

4. Choose **Zynq_FSBL** as the application template.



Templates

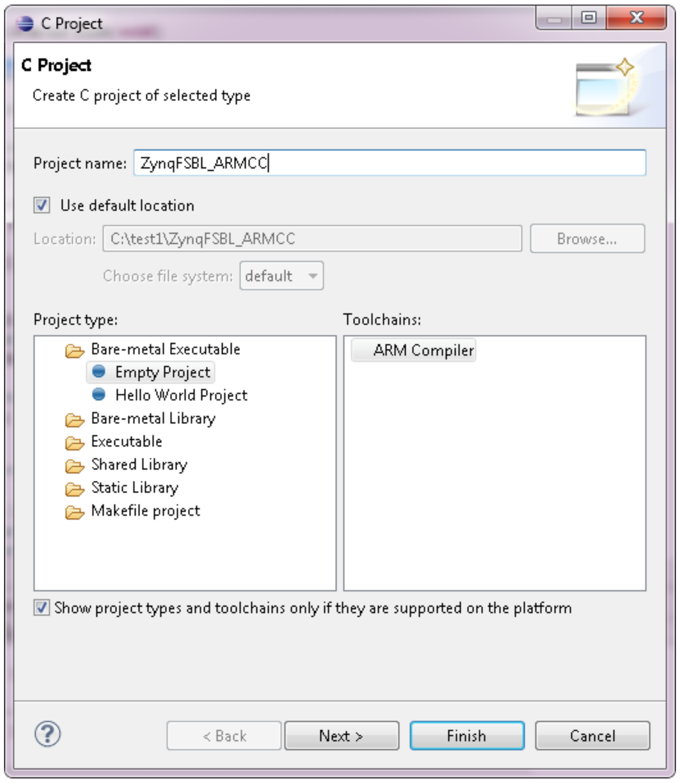
Create one of the available templates to generate a fully-functioning application project.

Available Templates:

Dhrystone
Empty Application
Hello World
lwIP Echo Server
Memory Tests
Peripheral Tests
[✓ Zynq FSBL]

First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream.

5. Click **Finish**.
6. Open Eclipse for DS-5.
7. Use the existing workspace, "DS5_Workspace."
8. Create a **Bare-metal Executable** > **Empty** application project named "ZynqFSBL." For more information, see the procedure in Step 3 - DS-5 Tools: Application Build.


C Project

C Project

Create C project of selected type

Project name: ZynqFSBL_ARMCC

☑ Use default location

Location: C:\test1\ZynqFSBL_ARMCC

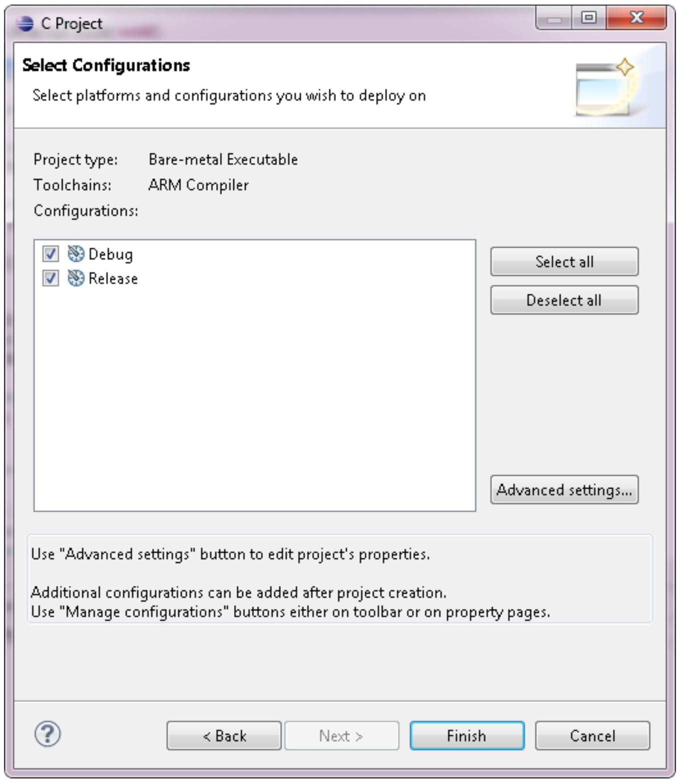
Choose file system: default

Project type:
- Bare-metal Executable
  - Empty Project
  - Hello World Project
- Bare-metal Library
- Executable
- Shared Library
- Static Library
- Makefile project

Toolchains:
- ARM Compiler

☑ Show project types and toolchains only if they are supported on the platform

9. Click **Finish**.



Select Configurations

Select platforms and configurations you wish to deploy on

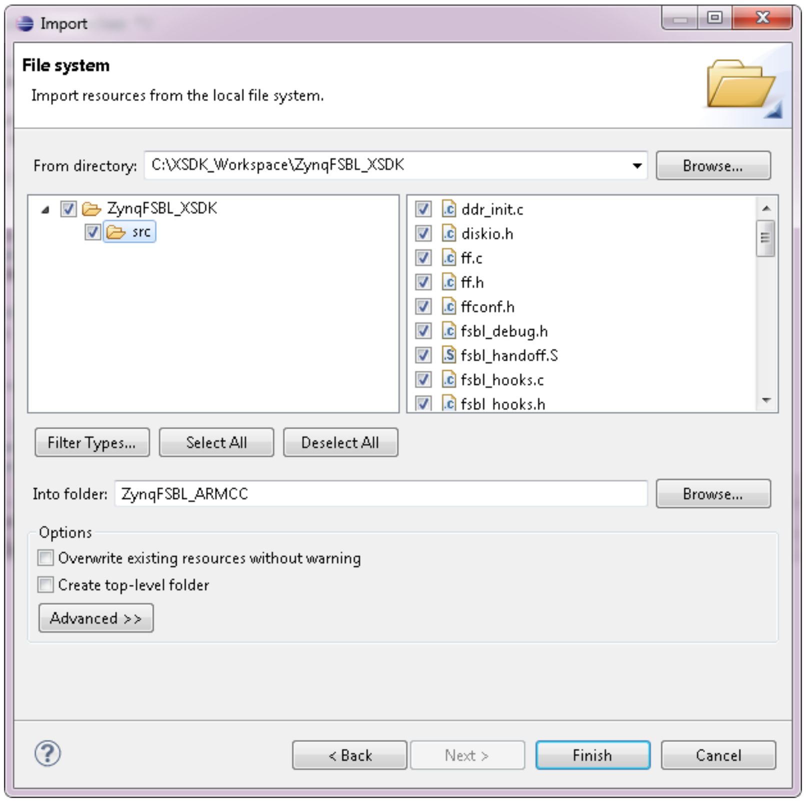
Project type: Bare-metal Executable
Toolchains: ARM Compiler
Configurations:

- ✓ Debug
- ✓ Release

Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

10. Import the files from "ZynqFSBL_XSDK" into this project.



File system

Import resources from the local file system.

From directory: C:\XSDK_Workspace\ZynqFSBL_XSDK

- ✓ ZynqFSBL_XSDK
  - ✓ src

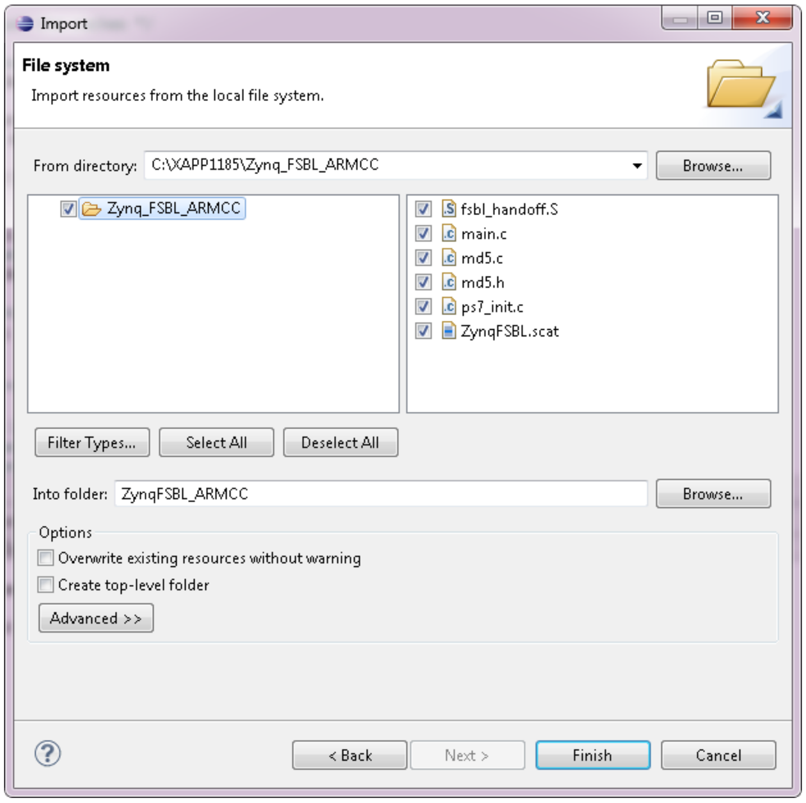
- ✓ ddr_init.c
- ✓ diskio.h
- ✓ ff.c
- ✓ ff.h
- ✓ ffconf.h
- ✓ fsbl_debug.h
- ✓ fsbl_handoff.S
- ✓ fsbl_hooks.c
- ✓ fsbl_hooks.h

Into folder: ZynqFSBL_ARMCC

Options

- ☐ Overwrite existing resources without warning
- ☐ Create top-level folder

11. Import the patched FSBL files from the support files location, specifically the files located in the `Zynq_FSBL_ARMCC` subfolder.



From directory: C:\XAPP1185\Zynq_FSBL_ARMCC

[✓] Zynq_FSBL_ARMCC

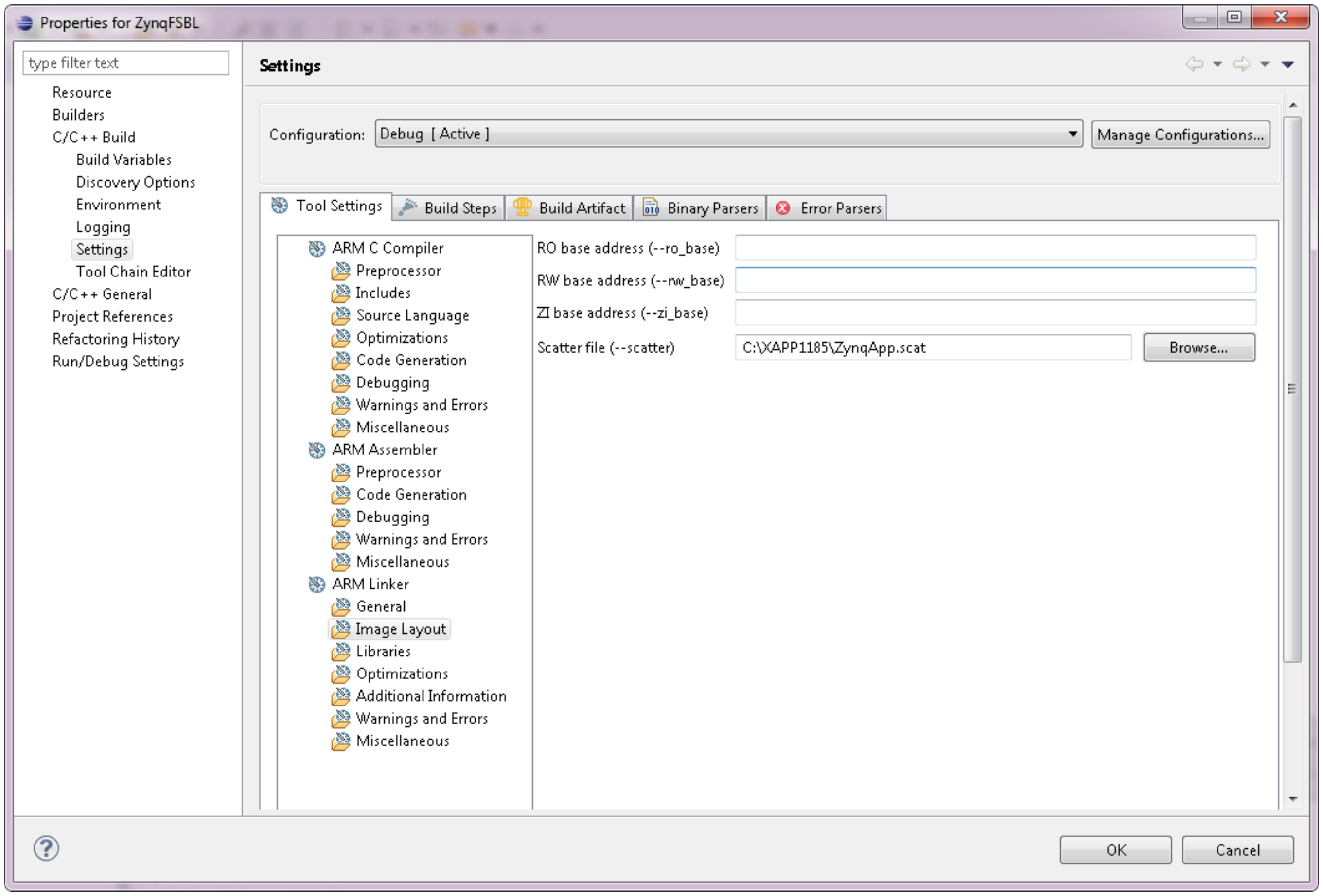
[✓] fsbl_handoff.S
[✓] main.c
[✓] md5.c
[✓] md5.h
[✓] ps7_init.c
[✓] ZynqFSBL.scat

Into folder: ZynqFSBL_ARMCC

Options
[ ] Overwrite existing resources without warning
[ ] Create top-level folder

12. Set the Compiler and Linker options as described in Step 3 - DS-5 Tools: Application Build.

13. Set the image layout file to `ZynqFSBL.scat`. This file is located at `..\XAPP1185\Zynq_FSBL_ARMCC`.



Configuration: Debug [Active]

RO base address (--ro_base)

RW base address (--rw_base)

ZI base address (--zi_base)

Scatter file (--scatter) C:\XAPP1185\ZynqApp.scat

14. Build the project “ZynqFSBL_ARMCC.”

The following illustration shows a sample, post-build log for reference.

```
===============================================================================

      Code (inc. data)   RO Data    RW Data    ZI Data      Debug

     72082      17986        450      11248      38620     124417   Grand Totals
     72082      17986        450       1796      38620     124417   ELF Image Totals (compressed)
     72082      17986        450       1796          0          0   ROM Totals

===============================================================================

    Total RO  Size (Code + RO Data)                72532 (  70.83kB)
    Total RW  Size (RW Data + ZI Data)             49868 (  48.70kB)
    Total ROM Size (Code + RO Data + RW Data)      74328 (  72.59kB)

===============================================================================
'Finished building target: ZynqFSBL.axf'
' '

**** Build Finished ****
```

You have now completed this step.

Note: The `ps7_init.c` file provide in `..\XAPP1185\ Zynq_FSBL_ARMCC` works only for ZC702_hw_platform. For other hardware platforms, you must manually edit this file to compile with ARMCC.

Step 5 - DS-5 Tools: Debugger Target Configuration for Zynq Device Custom Design

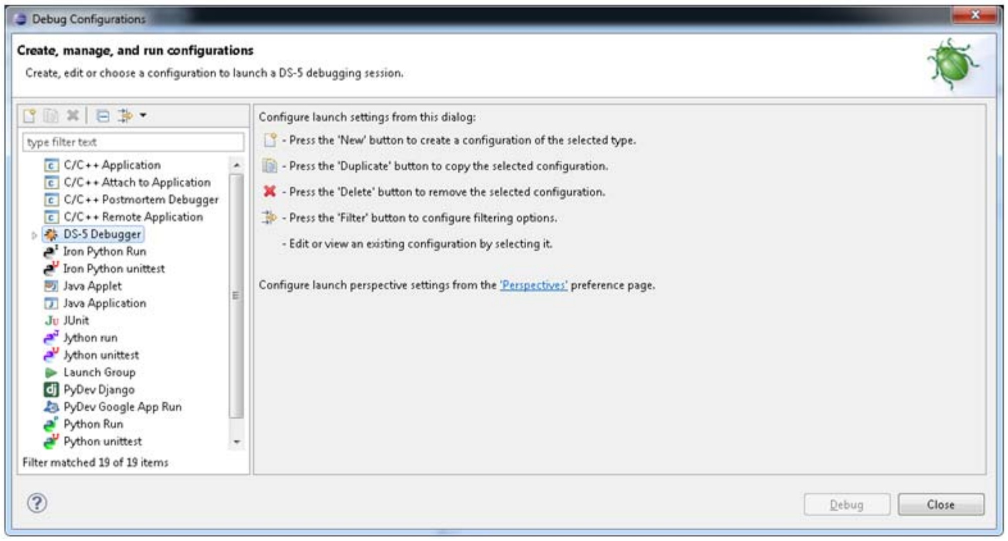
You can use the FSBL application created in Step 4 to initialize the target hardware. The following init script shows an example of using the `ZynqFSBL.axf` to initialize the target.

`zynq_init.ds`

```
loadfile {Workspace_loc:}\Zynq_FSBL_ARMCC\Debug\ZynqFSBL.axf
run
pause 1000
stop
```

Use the following steps to initialize the target hardware for a debug session:

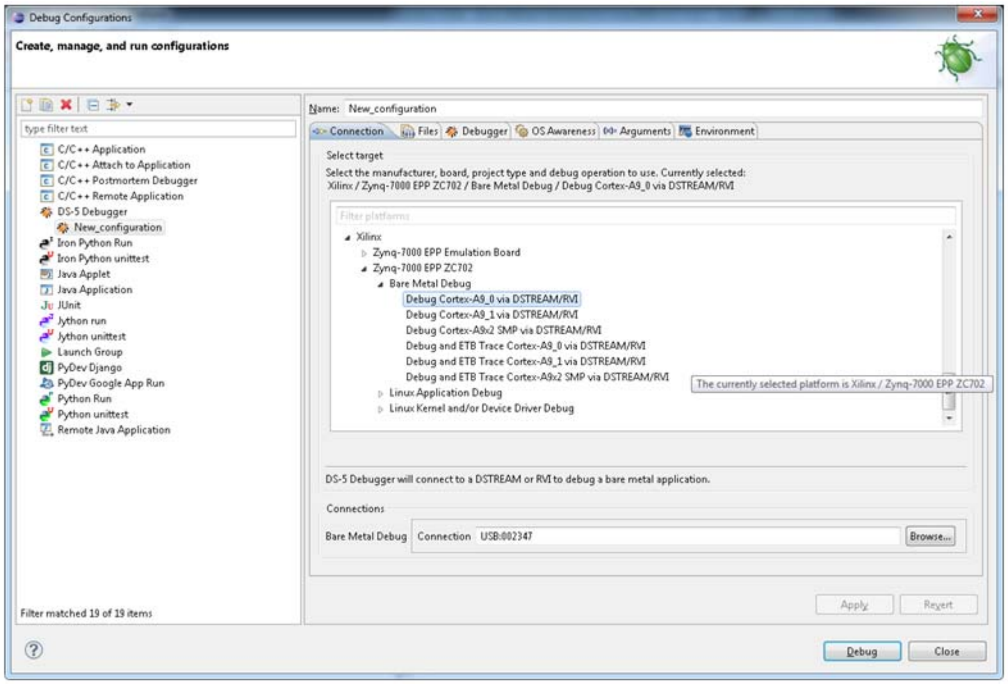
1. Right-click on the application that you want to debug and select **Debug Configurations**.
2. In the Debug Configurations window, double-click on **DS-5 Debugger** to create a new Debug Configuration.



Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

3. On the Connection tab, find “Zynq-7000 EPP ZC702” and select **Bare Metal Debug** > **Debug Cortex-A9 via DStream/RVI**.


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- DS-5 Debugger
  - New_configuration
- Iron Python Run
- Iron Python unittest
- Java Applet
- Java Application
- JUnit
- Jython run
- Jython unittest
- Launch Group
- PyDev Django
- PyDev Google App Run
- Python Run
- Python unittest
- Remote Java Application

Filter matched 19 of 19 items

Name: New_configuration

Connection | Files | Debugger | OS Awareness | Arguments | Environment

Select target

Select the manufacturer, board, project type and debug operation to use. Currently selected:
Xilinx / Zynq-7000 EPP ZC702 / Bare Metal Debug / Debug Cortex-A9_0 via DSTREAM/RVI

Filter platforms

- Xilinx
  - Zynq-7000 EPP Emulation Board
  - Zynq-7000 EPP ZC702
    - Bare Metal Debug
      - Debug Cortex-A9_0 via DSTREAM/RVI
      - Debug Cortex-A9_1 via DSTREAM/RVI
      - Debug Cortex-A9x2 SMP via DSTREAM/RVI
      - Debug and ETB Trace Cortex-A9_0 via DSTREAM/RVI
      - Debug and ETB Trace Cortex-A9_1 via DSTREAM/RVI
      - Debug and ETB Trace Cortex-A9x2 SMP via DSTREAM/RVI
    - Linux Application Debug
    - Linux Kernel and/or Device Driver Debug

The currently selected platform is Xilinx / Zynq-7000 EPP ZC702

DS-5 Debugger will connect to a DSTREAM or RVI to debug a bare metal application.

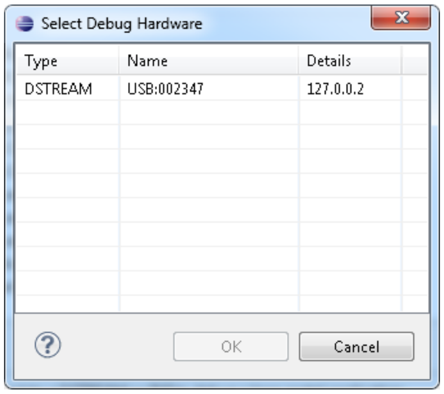
Connections

Bare Metal Debug | Connection: USB:002347 | Browse...

Apply | Revert

Debug | Close

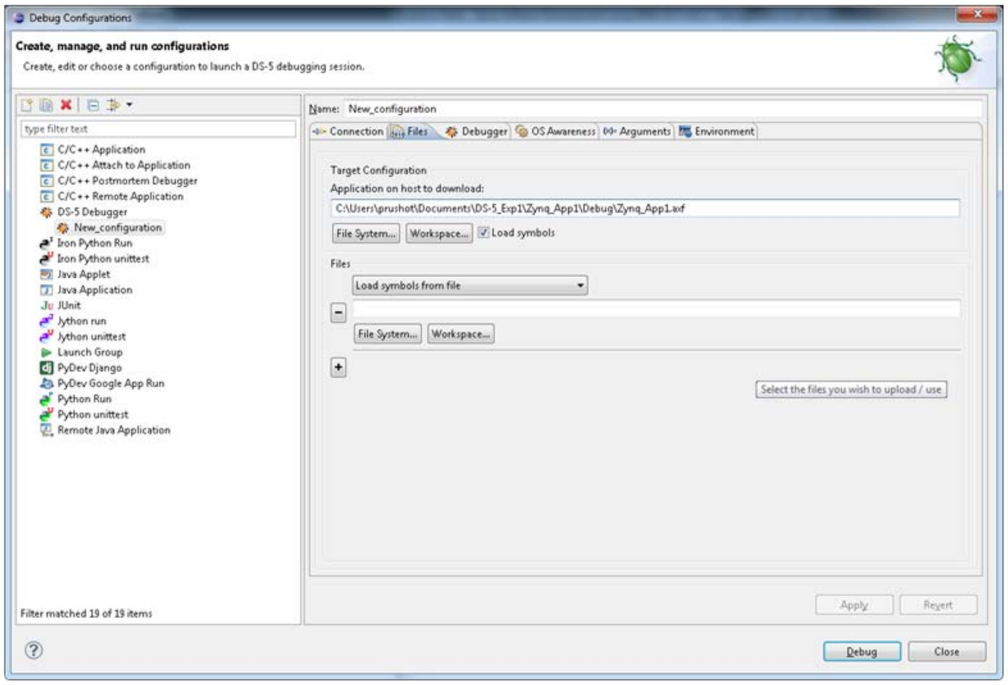
4. In the Connections area at the bottom of the dialog box, click the **Browse** button to the right of the Bare Metal Debug, Connection field. The application auto-detects the Dstream connection to the host machine. Select the appropriate connection (USB in this example), then click **OK**.


Select Debug Hardware

Type	Name	Details
DSTREAM	USB:002347	127.0.0.2

OK | Cancel

5. On the Files tab, select the application that you need to debug (for example, `zynq_app.axf`).



Name: New_configuration

Target Configuration

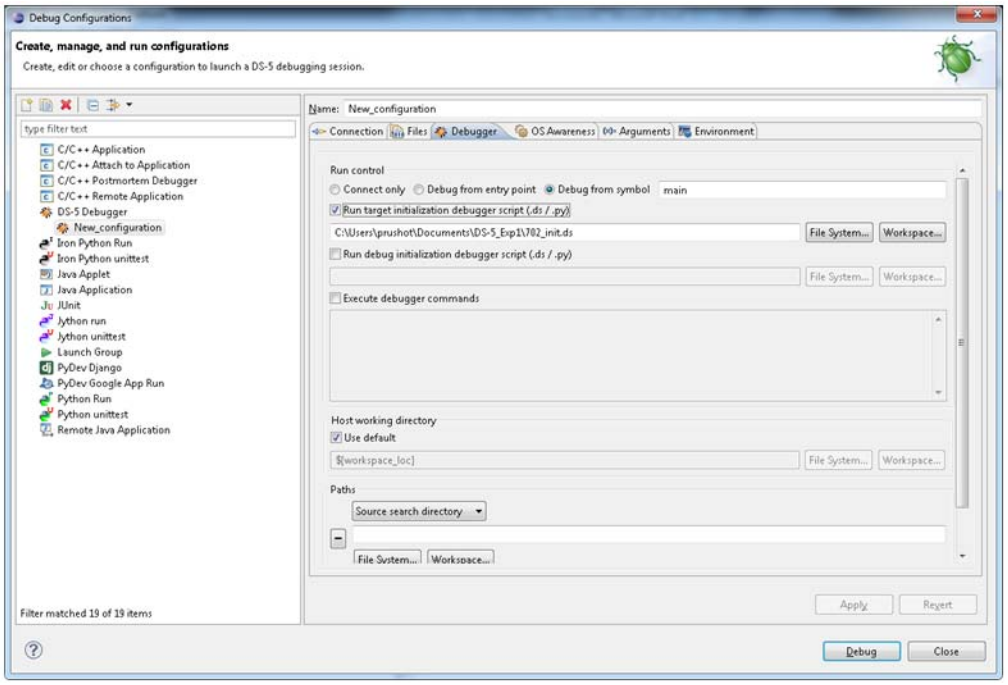
Application on host to download: C:\Users\prushot\Documents\DS-5_Exp1\Zynq_App1\Debug\Zynq_App1.axf

[✓] Load symbols

Files

Load symbols from file

6. On the Debugger tab, select the **Run target initialization debugger script (.ds/.py)** checkbox and select the `zynq_init.ds script` (example provided above).



Name: New_configuration

Run control

Connect only   Debug from entry point   [✓ Debug from symbol] main

[✓] Run target initialization debugger script (.ds / .py)

C:\Users\prushot\Documents\DS-5_Exp1\702_init.ds

[ ] Run debug initialization debugger script (.ds / .py)

[ ] Execute debugger commands

Host working directory

[✓] Use default

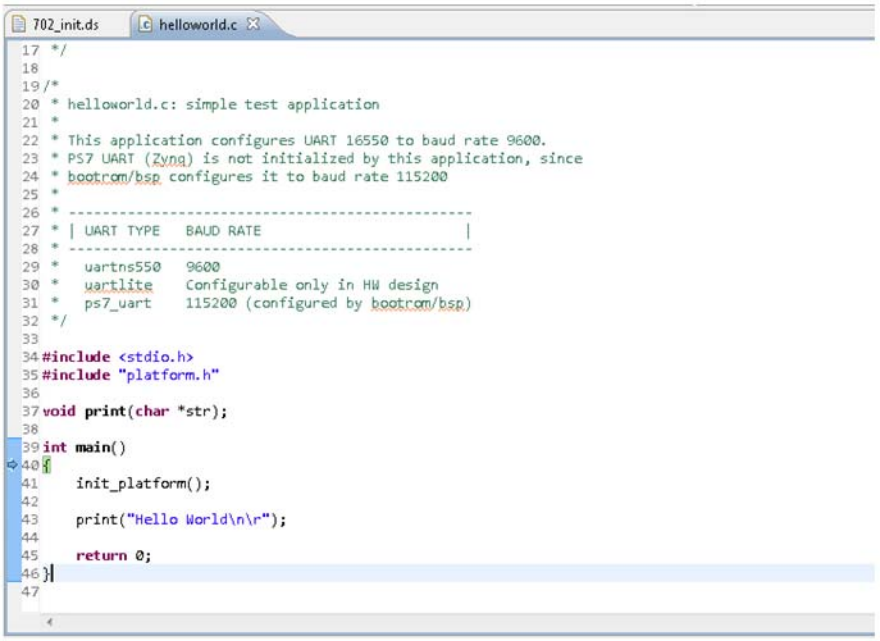
${workspace_loc}

Paths

Source search directory

7. Make sure the DStream cable is connected to the board. Power on DStream and the target.
8. Click the **Debug** button.

Note: The debugger loads FSBL in OCM and runs it. The FSBL then initializes the target hardware. The debugger waits one second for FSBL to complete initialization of the target and stops the target to load the application with a breakpoint at main().



```
 */

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"

void print(char *str);

int main()
{
    init_platform();

    print("Hello World\n\r");

    return 0;
}

```

This concludes the steps necessary to set up a debug session. For successive debug sessions, click the **Debug** button.

Automating Step 1 Using Scripts

Advanced users can automate Step 1 of this application note with the included script files. The batch file to automate step 1, `build_bsp_fsbl_app.bat`, is located at `..\XAPP1185\xsdk_build\build_bsp_fsbl_app.bat`.

The location of the `system.xml` file has to be passed as an argument to run this batch file, as shown in the following example, which generates the ARMCC version of BSP and FSBL for the ZC702 board. The command assumes the design files `XAPP1185.zip` is unzipped to C:\.

```
C:\XAPP1185\xsdk_build>build_bsp_fsbl_app.bat ZC702_hw_platform
```

This command takes the `system.xml` file from ZC702_hw_platfrom, which is available as part of this document. The batch file produces three folders: `Zynq_FSBL`, `ps7_cortex_a9`, and the `hello_world` project folder. You can import these files directly into the ARM DS-5 project as described in Step 2 - DS-5 Tools: BSP Import and Build.

References

1. *Zynq-7000 All Programmable SoC: Concepts, Tools, and Techniques* (UG873)
2. Xilinx video tutorial, "Zynq Bare Metal Application Development using Xilinx SDK"
3. Xilinx video tutorial, "Heterogeneous Multicore Debugging with Xilinx SDK"

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
11/18/13	1.0	Initial Xilinx release.

Notice of Disclaimer