XAPP1185 (v2.0) May 6, 2014

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

Authors: 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 software for the Xilinx® Zynq®-7000 All Programmable SoC, which is based on the ARM Cortex™-A9 processor.

- Introduction
- Step 1: Run the Project Auto Creation Batch Script
- Step 2: DS-5 Tools, Import and Build the Projects
- Step 3: DS-5 Tools, Develop/Debug Software on Zynq Device Hardware Target

The steps in the Appendix section of this document are provided in case you would like to understand what the batch file does or if you have a reason to perform the tasks accomplished by the batch file manually.

Introduction

This document assumes:

- A basic understanding of the architecture, boot flow, and associated Xilinx design tools for the Zynq-7000 product family.
- A basic understanding of the ARM DS-5 tool chain.
- The development host operating system is Windows.

Software Requirements

To complete the procedures in this application note, you must have the following host-based software tools installed:

- Xilinx Software Development Kit (SDK), which you can download from www.xilinx.com/support/download/index.htm (2013.3 or 2013.4)
- ARM Development Studio 5 (DS-5 Professional) tool suite, which you can download from ds.arm.com/downloads/.

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

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

Hardware Requirements

Debug Hardware

You must have one of the following debug units:

- uLINKpro D/ uLINKpro - Debug/Debug and Trace unit
- ARM DSTREAM™ High Performance Debug and Trace unit

Boards

- ZC702
- Any Zynq hardware fitted with a suitable JTAG header

Reference Designs

The reference design files for this application note are available in a ZIP file at:
https://secure.xilinx.com/webreg/clickthrough.do?cid=351623

The ZIP file contains two folders:

- `xapp1185\AutoScript_2013.3`
- `xapp1185\AutoScript_2013.4`

Use the reference design files that corresponds to your Vivado® Design Suite version.

Note: At the time of this document release, version 2014.1 is not supported.

The design files include the Windows batch file, `ProjectAutoCreation.bat,` which lets you generate the BSP, FSBL, and the Hello World application project. Instructions on use of the BAT file are provided later in this document. The table describes the reference design files in detail.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer Name(s)	Simon George; Prushothaman Palanichamy
Target Devices (Stepping Level, ES, Production, Speed Grades)	Zynq-7000 device
Source Code Provided	Y
Source Code Format	C, Assembly, Batch Script
Design Uses Code/IP from Existing Application Note, Reference Designs, Third Party, or CORE Generator™ Software	N
Simulation	
Functional Simulation Performed	N/A
Timing Simulation Performed	N/A
Testbench Used for Functional and Timing Simulations	N/A
Testbench Format	N/A
Simulator Software/Version Used	N/A
SPICE/IBIS Simulations	N/A
Implementation	
Synthesis Software Tools/Version Used	N/A
Implementation Software Tools/Versions Used	SDK 2013.3, SDK 2013.4. ARM DS-5 Professional v5.17.
Static Timing Analysis Performed	N/A
Hardware Verification	N/A
Hardware Verified	Y
Hardware Platform Used for Verification	ZC702

Step 1: Run the Project Auto Creation Batch Script

Perform the following steps to run the batch file and generate the Standalone BSP, FSBL, Hello World Application, and Support File projects:

1. Open a DS-5 command prompt.

As shown in the figure below:

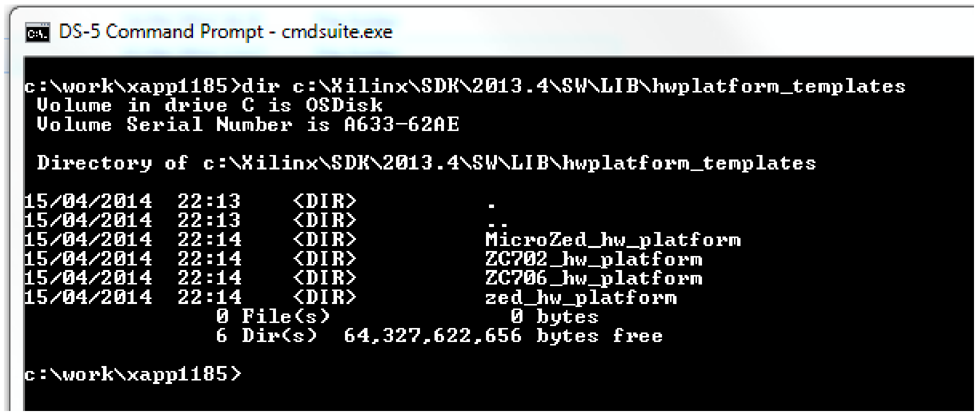
2. Change the directory to `<xapp1185_directory>\AutoScript_<version>`.
3. Run the batch file `ProjectAutoCreation.bat` with three arguments:

   Argument #1: Xilinx Tools Install Base

   This is the path to the Xilinx tool installation and specifically looks for the `settings32|64.bat` file. For SDK installations only, this is in the XSDK base install directory. For Vivado Design Suite with SDK installations, this is in the Vivado Design Suite base install directory.

   Argument #2: Hardware XML Directory

   Location of the `system.xml` hardware description file for the processor configuration against which the software platform is to be built. Hardware templates are provided in the tools install for the development platforms shown in the figure below. The templates can be used in advance of custom hardware availability.



DS-5 Command Prompt - cmdsuite.exe

```
c:\work\xapp1185>dir c:\Xilinx\SDK\2013.4\SW\LIB\hwplatform_templates
 Volume in drive C is OSDisk
 Volume Serial Number is A633-62AE

 Directory of c:\Xilinx\SDK\2013.4\SW\LIB\hwplatform_templates

15/04/2014  22:13    <DIR>          .
15/04/2014  22:13    <DIR>          ..
15/04/2014  22:14    <DIR>          MicroZed_hw_platform
15/04/2014  22:14    <DIR>          ZC702_hw_platform
15/04/2014  22:14    <DIR>          ZC706_hw_platform
15/04/2014  22:14    <DIR>          zed_hw_platform
               0 File(s)              0 bytes
               6 Dir(s)  64,327,622,656 bytes free

c:\work\xapp1185>
```

Figure 1: **Hardware Templates for Development Platforms**

   Argument #3: Output Directory

   Location at which to create output files and folders.

   Example Command line (one line):

```
ProjectAutoCreation.bat c:\Xilinx\Vivado\2013.4\
c:\Xilinx\SDK\2013.4\sw\lib\hwplatform_templates\ZC702_hw_platform
..\Output
```

4. On successful script completion, the following message appears:

```
Successfully created DS-5 Projects for Import!!!
```

In addition, four DS-5 projects to import are created in the defined output location, as shown in the figure below.

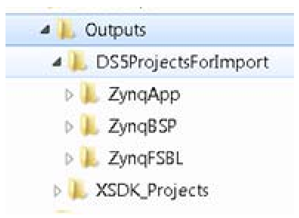


Figure 2: **Project Folders Created by Running the Batch File**

Step 2: DS-5 Tools, Import and Build the Projects

1. Invoke **Eclipse for DS-5** and open a new or existing workspace.
2. Import the four projects created in Step 1:
   a. In the DS-5 tool, click **File > Import** to open the Import wizard.

   In the Import wizard Select screen:

   b. Select **General** > **Existing Projects into Workspace.**

   c. Click **Next.**

   Figure 3 and Figure 4, below, illustrate steps a through c, above.

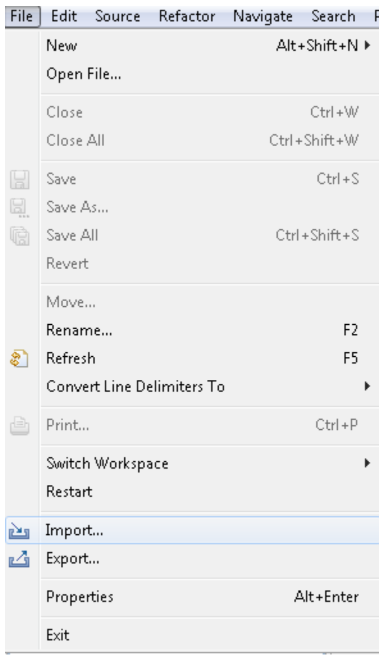


Figure 3: **Import Option in File Menu**

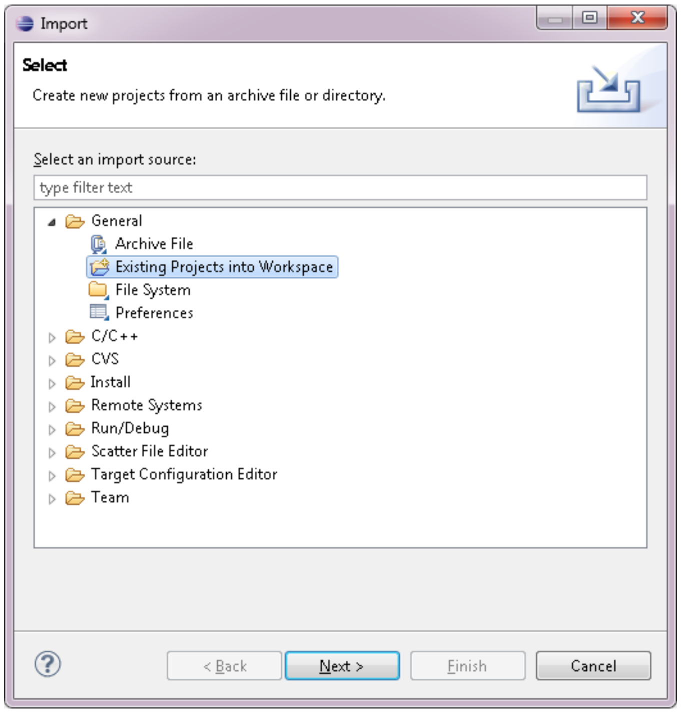


Figure 4: **Import Wizard *Select* Screen**

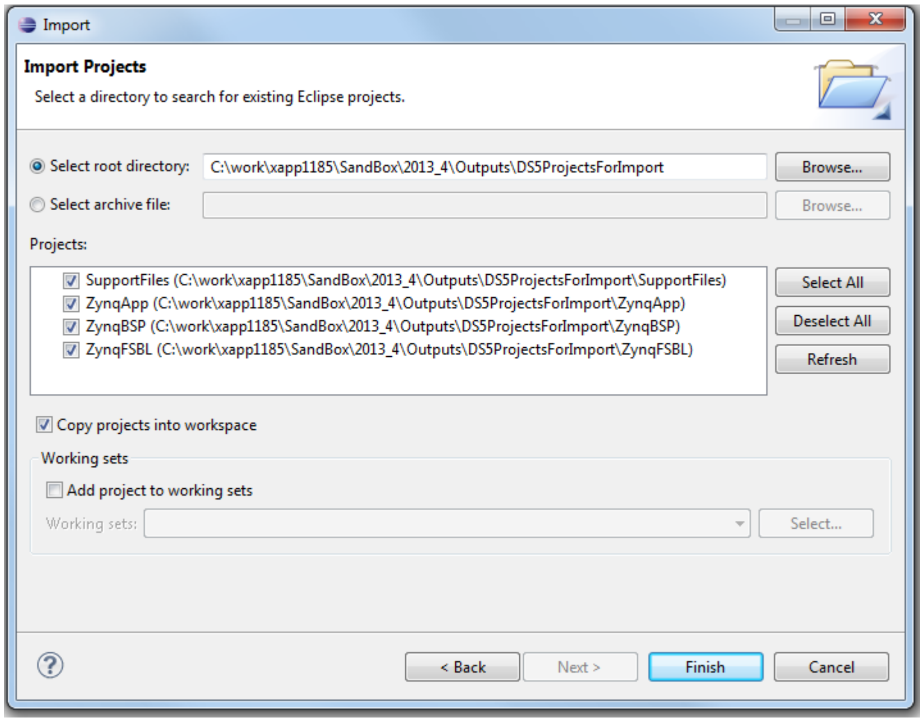
In the Import wizard Import Projects screen, shown in Figure 5, below:

d. In the Select Root Directory field, browse to the output directory specified in Step 1.

e. Select all four folders.

f. Check **Copy projects into workspace** (as shown in the figure below).

g. Click **Finish**.



Select root directory: C:\work\xapp1185\SandBox\2013_4\Outputs\DS5ProjectsForImport

Select archive file:

Projects:

- [✓] SupportFiles (C:\work\xapp1185\SandBox\2013_4\Outputs\DS5ProjectsForImport\SupportFiles)
- [✓] ZynqApp (C:\work\xapp1185\SandBox\2013_4\Outputs\DS5ProjectsForImport\ZynqApp)
- [✓] ZynqBSP (C:\work\xapp1185\SandBox\2013_4\Outputs\DS5ProjectsForImport\ZynqBSP)
- [✓] ZynqFSBL (C:\work\xapp1185\SandBox\2013_4\Outputs\DS5ProjectsForImport\ZynqFSBL)

[✓] Copy projects into workspace

[ ] Add project to working sets

Working sets:

Figure 5: **Import Wizard, *Import Projects* Screen**

h. Click **Project > Build All** (or use the shortcut **Ctrl+B**) to build all imported projects. Project dependencies in the environment files ensure the appropriate build sequence.

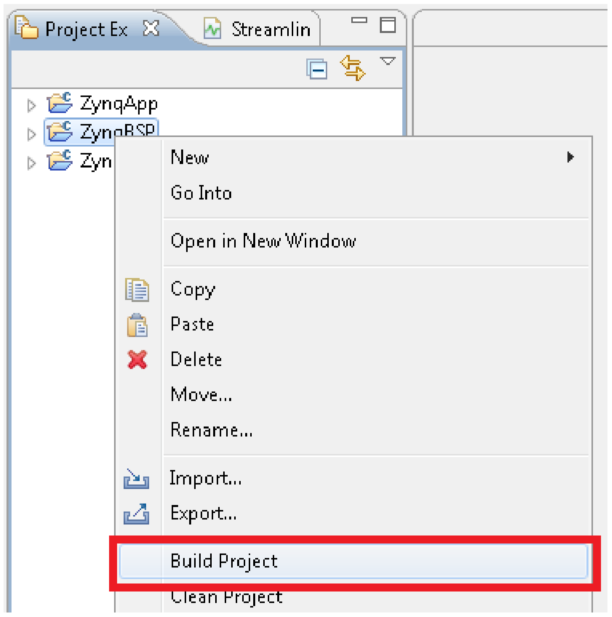


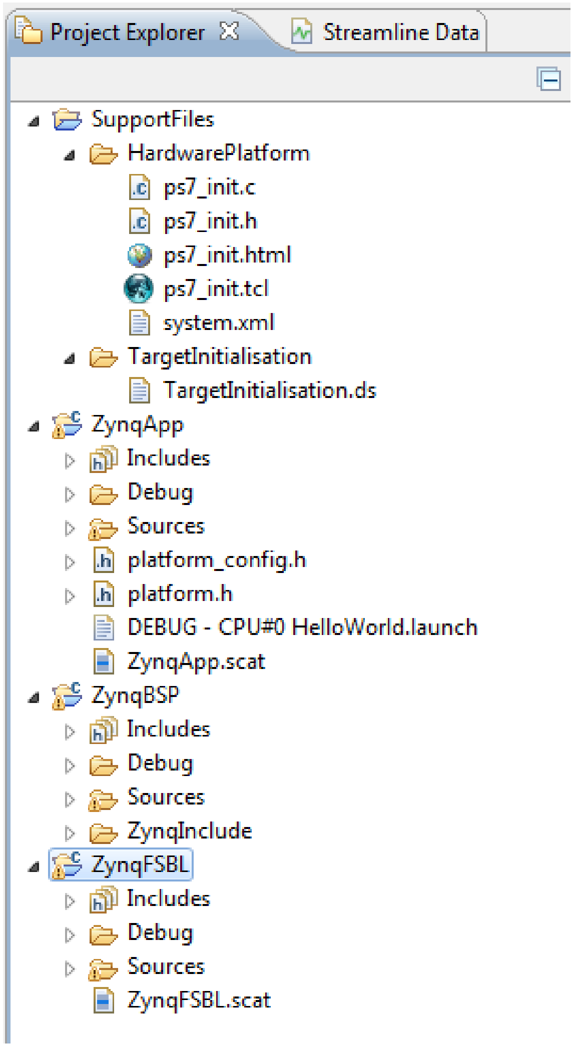
Figure 6: **Right-click Option to Build Projects (start with ZynqBSP)**

Note: Building the ZynqBSP project creates a custom library, `libZynqBSP.a`, for the target hardware. This library must be linked while building the ZynqAPP or ZynqFSBL project.

The following three files are created:

```
ZynqFSBL.axf
ZynqAPP.axf
libZynqBSP.a
```

The figure below shows the imported project folders and first-level content.



```
SupportFiles
  HardwarePlatform
    ps7_init.c
    ps7_init.h
    ps7_init.html
    ps7_init.tcl
    system.xml
  TargetInitialisation
    TargetInitialisation.ds
ZynqApp
  Includes
  Debug
  Sources
  platform_config.h
  platform.h
  DEBUG - CPU#0 HelloWorld.launch
  ZynqApp.scat
ZynqBSP
  Includes
  Debug
  Sources
  ZynqInclude
ZynqFSBL
  Includes
  Debug
  Sources
  ZynqFSBL.scat
```

Figure 7: **Imported Project Folders Showing First-Level Content**

Step 3: DS-5 Tools, Develop/Debug Software on Zynq Device Hardware Target

The FSBL application (linked into OCM) is used here to initialize the target hardware before running the primary application being developed/debugged.

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

1. Configure the JTAG chain as appropriate for your Zynq device board configuration. For example, on the ZC702 board, the JTAG chain must be configured in Cascade mode and mapped to J58. This is achieved by setting (SW10) as shown in shown in Figure 8, below.
2. Make sure your debug cable is connected to the board.
3. Power on the debug unit and target hardware, as appropriate.

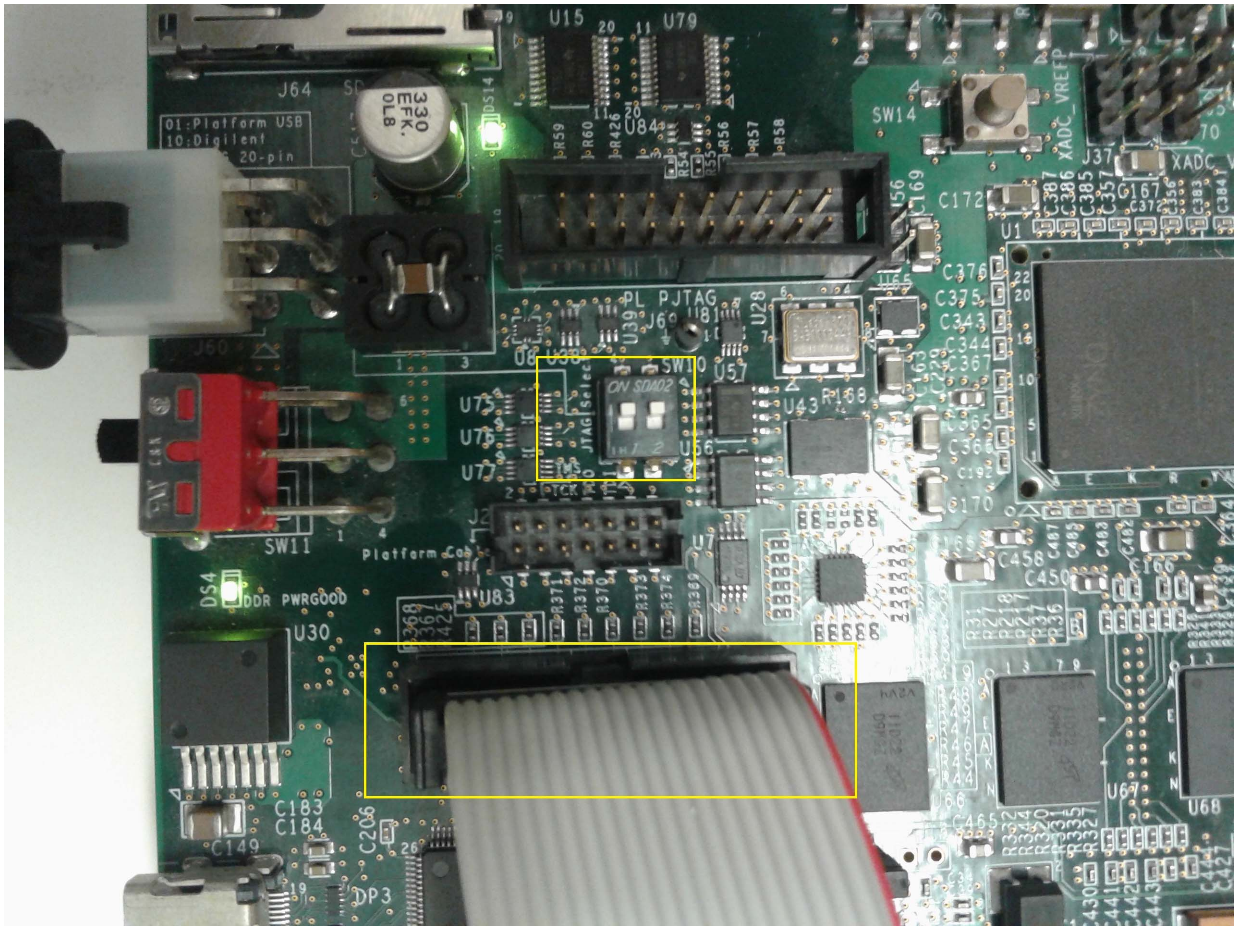
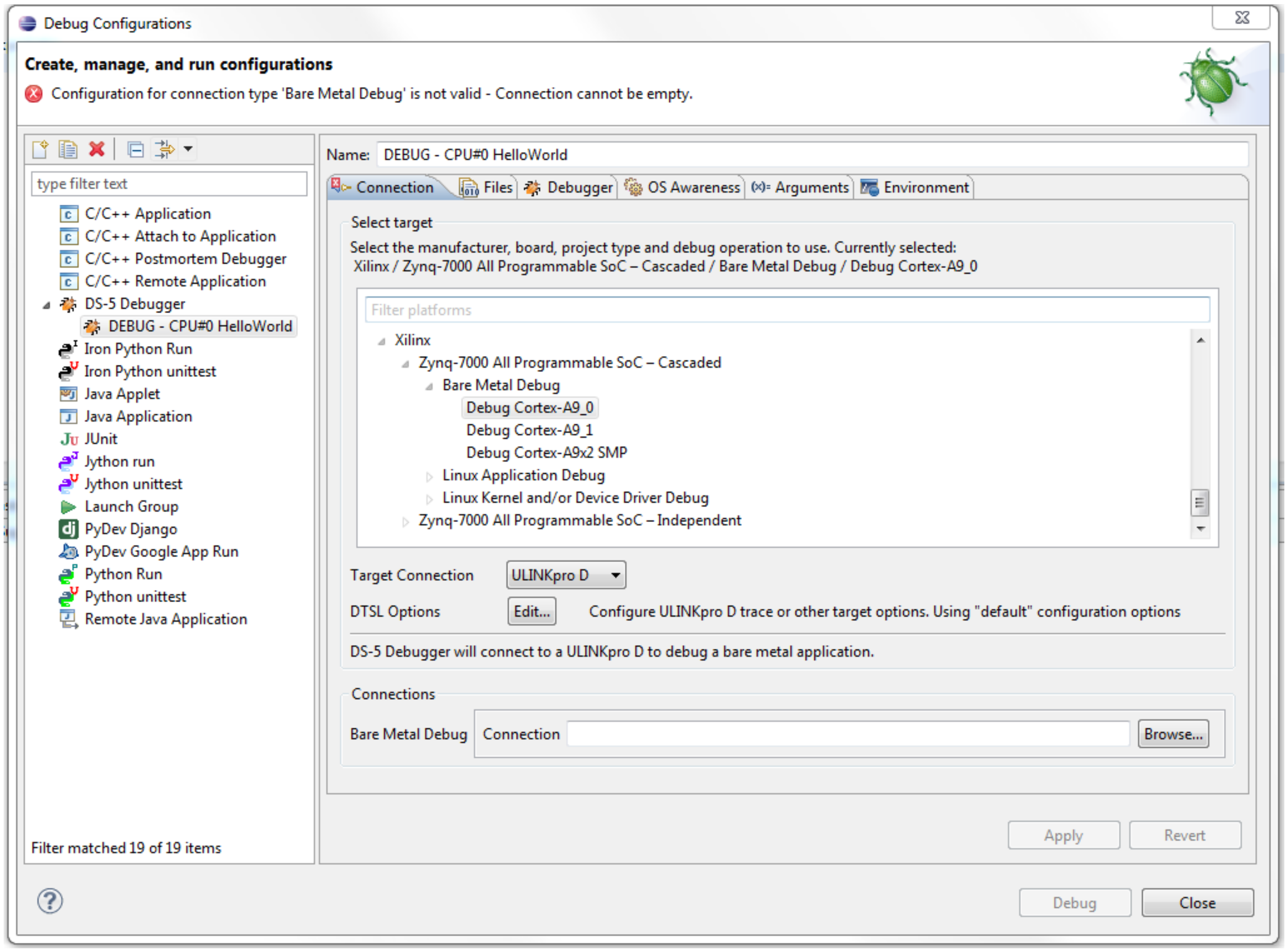


Figure 8: **Debug Setup (on ZC702 Board)**

4. Click the application (ZynqApp) you want to debug and select **Debug Configurations**.
5. In the Debug Configurations window (shown in Figure 9, below) expand the **DS-5 Debugger** entry and select the launch configuration provided for your reference. This assumes you wish to debug code built for CPU#0 on a target configured in Cascade JTAG mode.


Debug Configurations

Create, manage, and run configurations

Configuration for connection type 'Bare Metal Debug' is not valid - Connection cannot be empty.

Name: DEBUG - CPU#0 HelloWorld

Select target

Select the manufacturer, board, project type and debug operation to use. Currently selected:
Xilinx / Zynq-7000 All Programmable SoC – Cascaded / Bare Metal Debug / Debug Cortex-A9_0

Target Connection: ULINKpro D

DTSL Options: Configure ULINKpro D trace or other target options. Using "default" configuration options

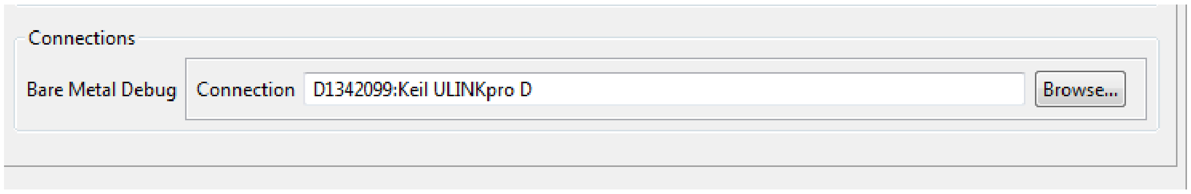
DS-5 Debugger will connect to a ULINKpro D to debug a bare metal application.

Connections

Bare Metal Debug Connection

Figure 9: **Debug Configuration Window, Connection Tab**

6. You must still select the target probe. Click **Browse** in the Connections section of the Debug Configuration tab (shown in Figure 10, below) and select the probe that you have attached to the host PC.

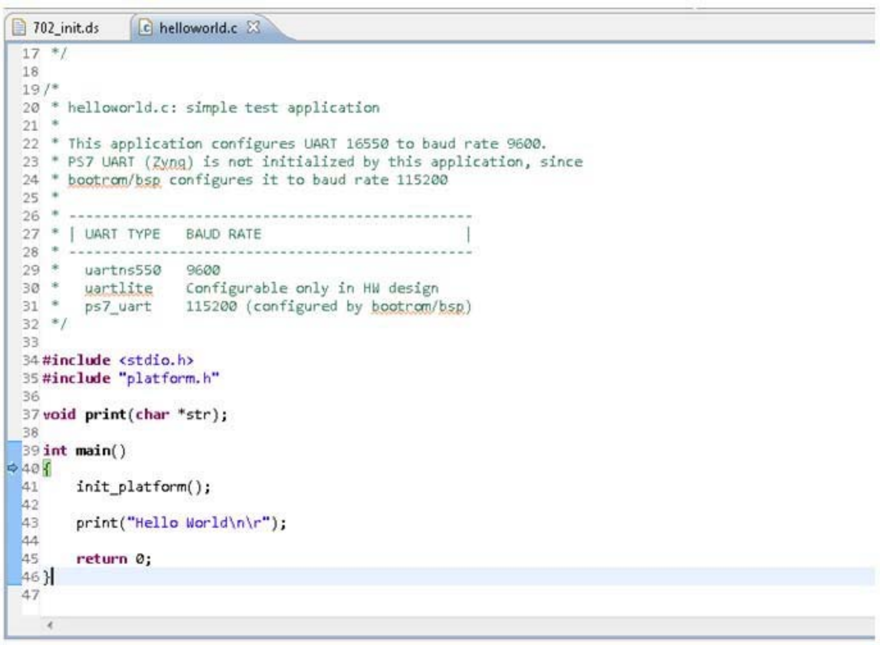

Connections

Bare Metal Debug Connection D1342099:Keil ULINKpro D

Figure 10: **Debug Configurations Window: Browse Option to Select Probe**

7. Debug the application on the target. As shown in Figure 11, the debugger stops at the entry point to the debugged application.

   This, along with other behavior, is defined through selectable options available in the launch configuration. Make the selections appropriate to your requirements. Settings and selections in this document are provided as examples only.



```
 */

/*
 * 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;
}
```

Figure 11: **Debugging Hello World Example**

This concludes the steps necessary to set up a debug session.

Appendix

The following steps summarize how to build the BSP, FSBL, and Hello World application without using a batch file. There are three major sections:

- Step 1: Xilinx SDK, Create the Standalone Board Support Package for custom hardware design
- Step 2: DS-5 Tools, Import and Build the BSP
- Step 3: DS-5 Tools, Create a DS-5 Application Project for the Zynq Device

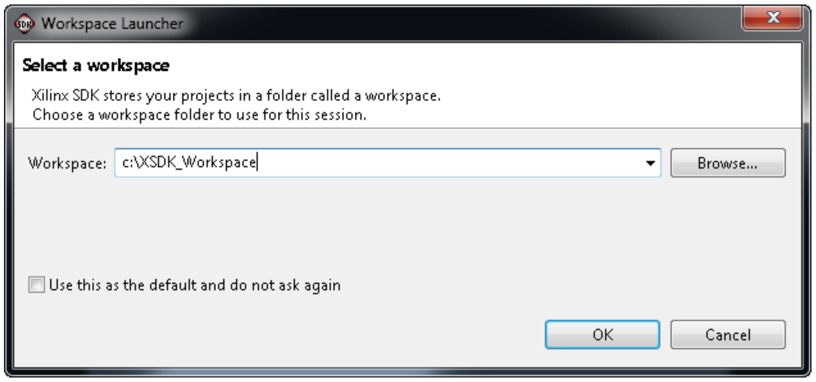
Step 1: Xilinx SDK, Create the Standalone Board Support Package

Xilinx SDK dynamically assembles a customized BSP based on the selected hardware design, whether that is a customized design imported from the Vivado Design Suite or a pre-configured platform. This assembled BSP includes a collection of CPU complex-specific configuration and startup code, along with platform address and configuration information for both a 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 the Library Generator (libgen) utility, which is an integral part of the Xilinx SDK environment.

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`), as shown in the figure below.



Workspace Launcher

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

Figure 12: **SDK Workspace Launcher Window**

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

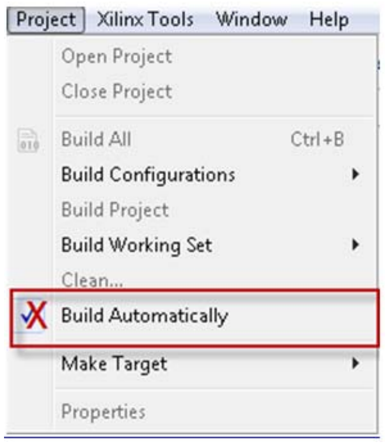


Figure 13: **Check Box to Toggle "Build Automatically" Option On/Off**

Steps c and d, below, are necessary only If you are using Vivado Design Suite version 2013.3 :

c. As shown in Figure 14, select **Xilinx Tools > Repositories** and click **New** on the Local Repositories box.

d. Browse to the `\AutoScript_2013.3\Templates\XSDK_Patches` folder in the application note design files and add the patched Standalone BSP as new repository.

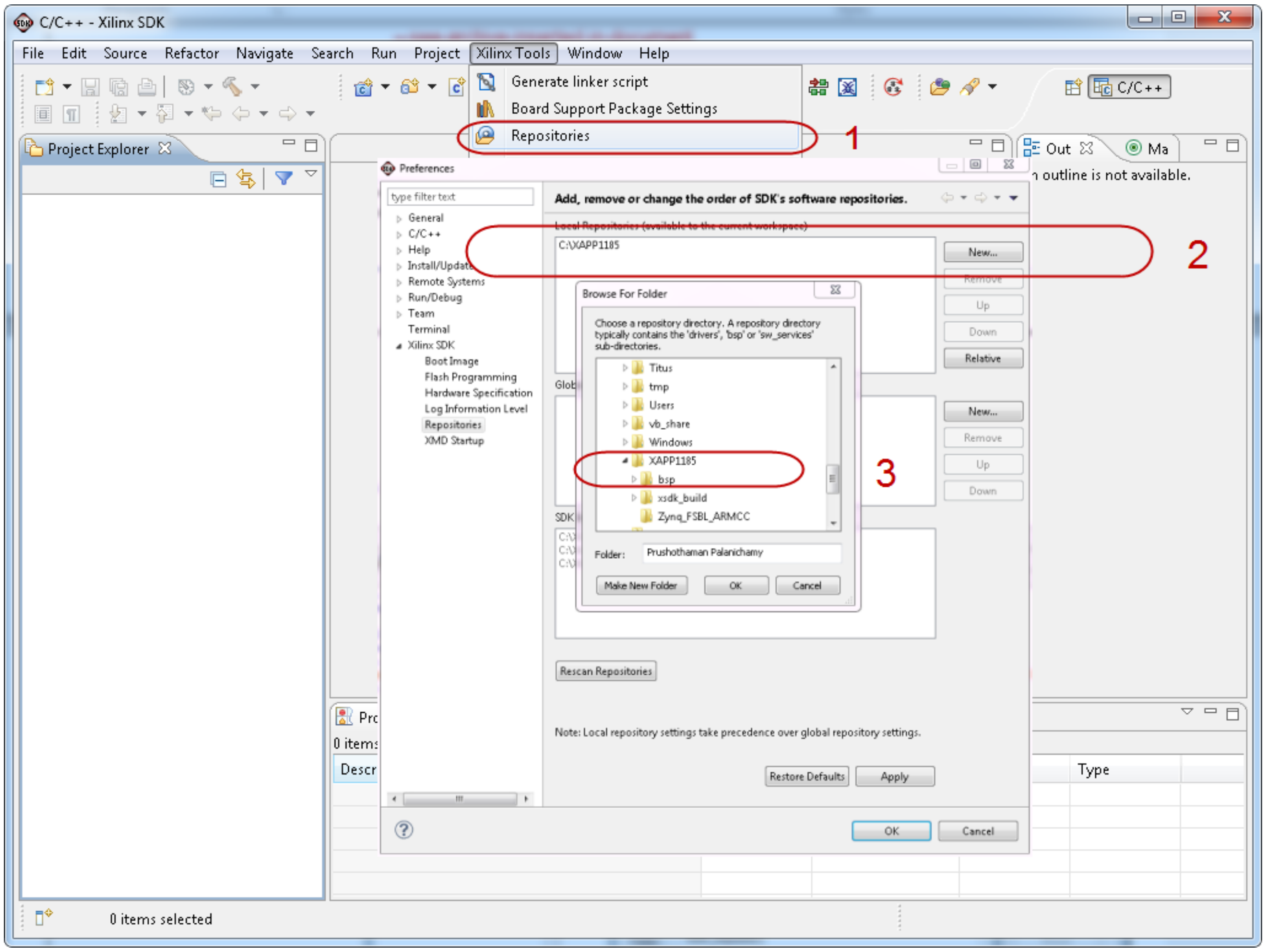


Figure 14: **Adding Patched BSP Repository for Projects 2013.3 and Earlier**

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 might 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,** as shown in the figure below.

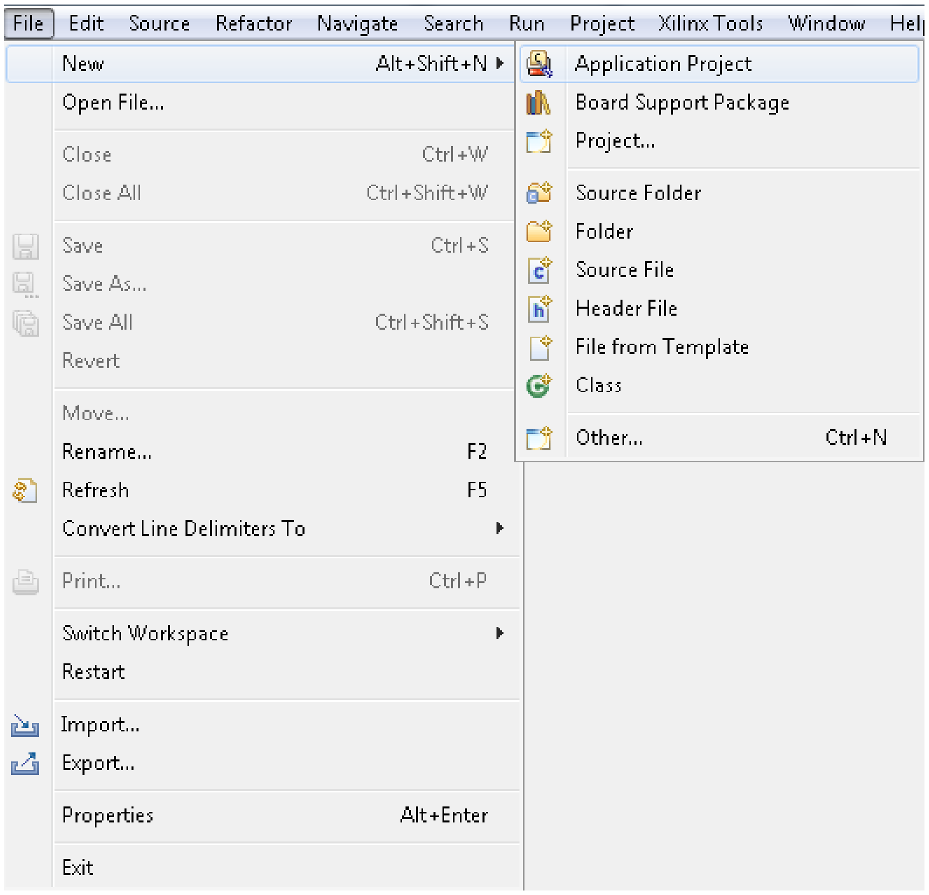


Figure 15: **New Application Project Selection from File Menu**

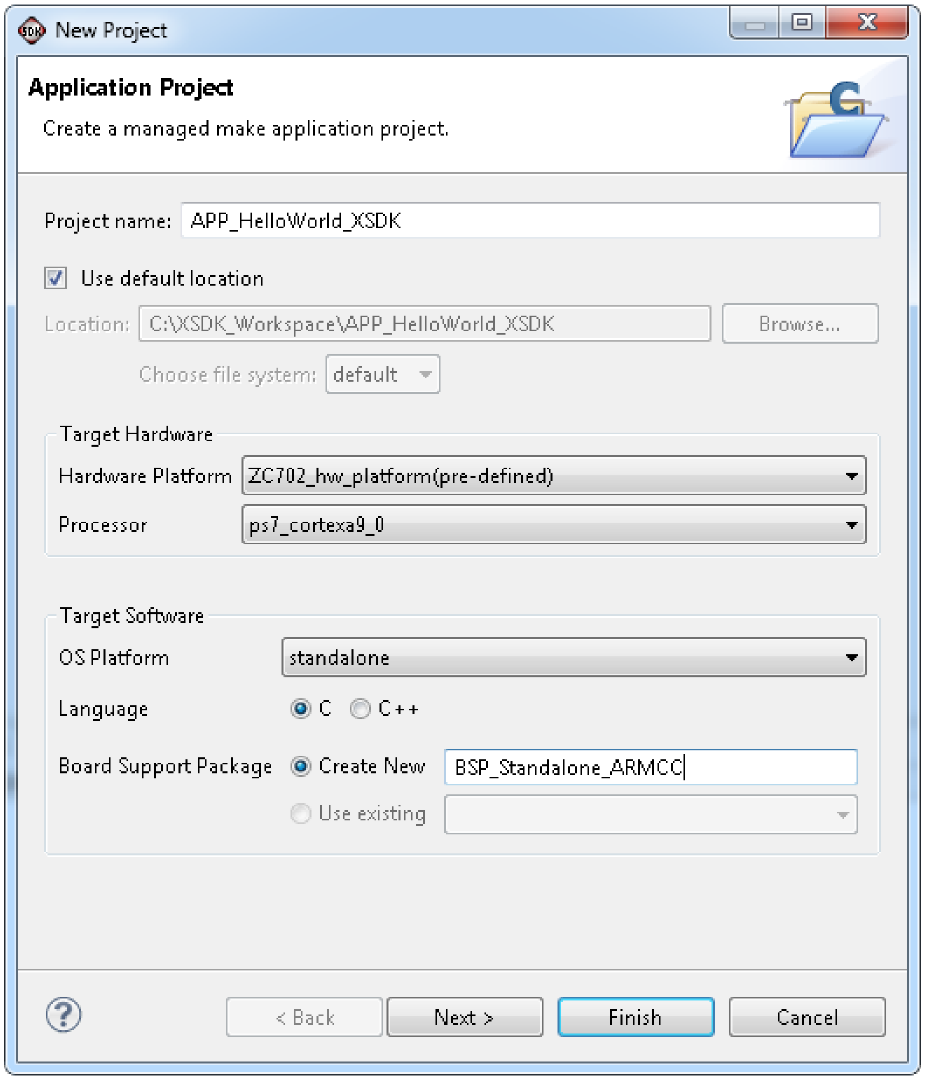
b. In the Application Project screen make the following settings:

- 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 figure. (Xilinx SDK uses the Xilinx-distributed GCC to build the BSP source by default, but you modify the tool chain options later in this procedure.)



New Project

Application Project

Create a managed make application project.

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

Use existing

Figure 16: **Application Project Screen**

- Click **Next**.
- Set the application template, shown in the figure below.

  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**.

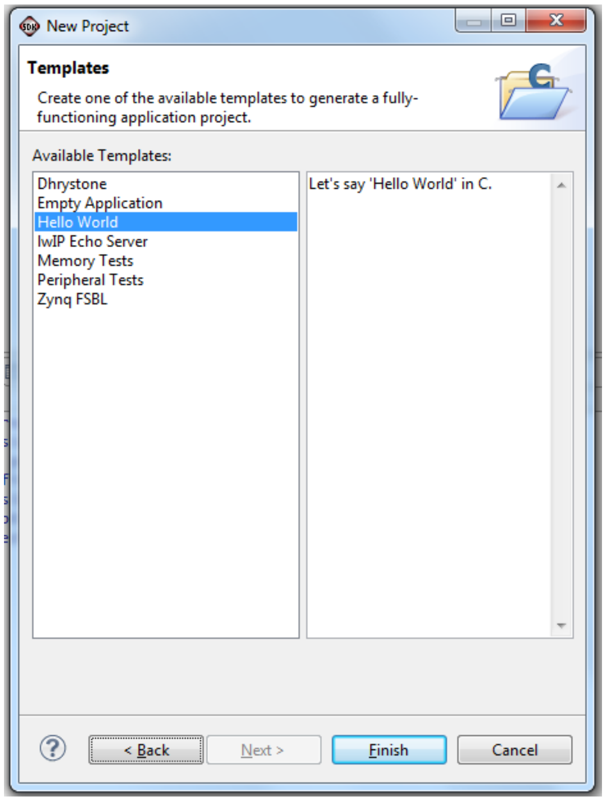


Figure 17: **Application Template Selections**

The tools create the files shown in the following figure:

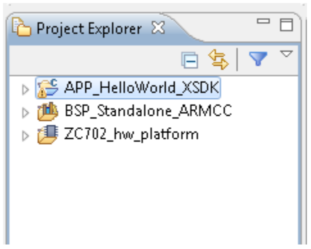


Figure 18: **Tool-Generated Files**

c. Right-click the BSP project, `BSP_Standalone_ARMCC`, and select **Board Support Package Settings,** as shown in the figure below.

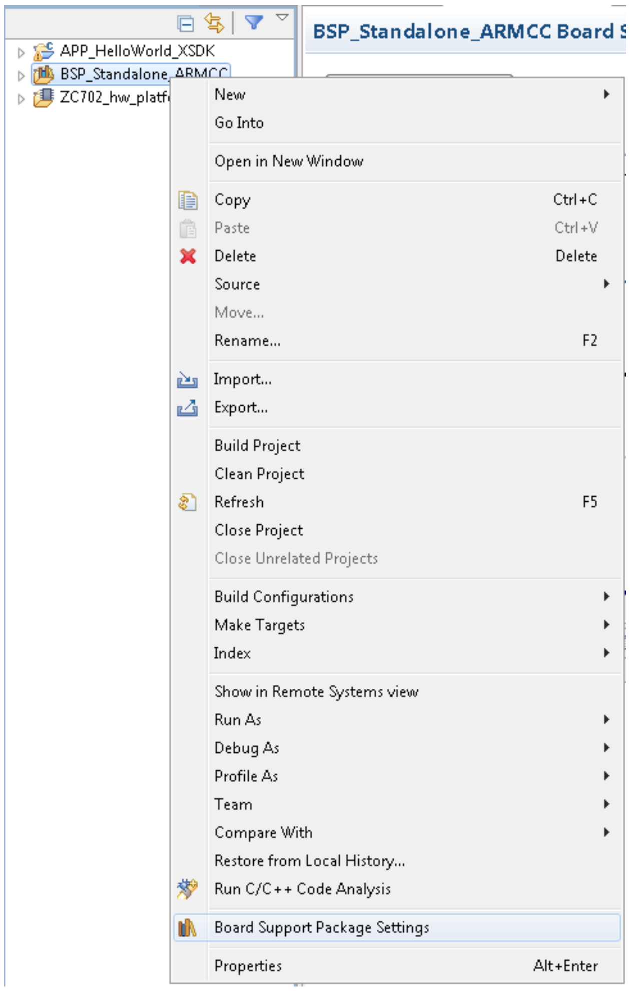
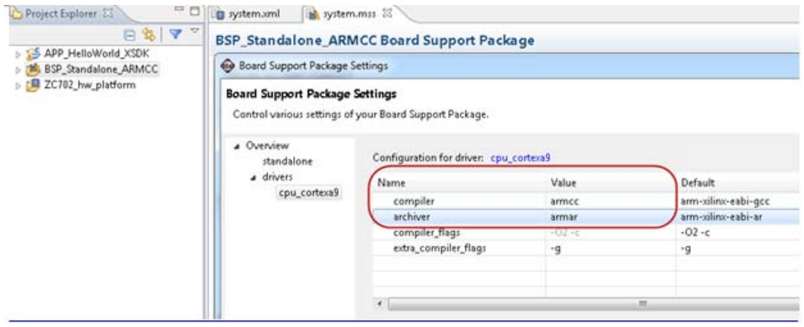


Figure 19: **BSP Settings Menu Option**

d. Navigate to **drivers** > **cpu_cortexa9** and change the following parameters (also shown in the figure below).

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

Figure 20: **BSP Driver Settings**

e. Right-click the BSP project and select **Build Project** (see figure).

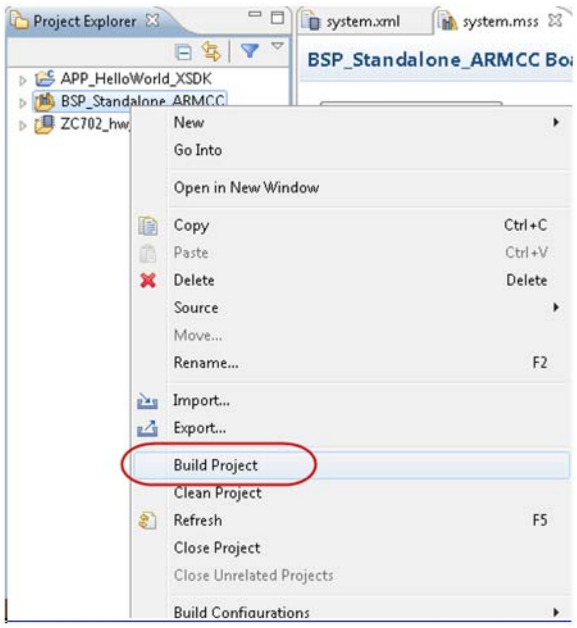


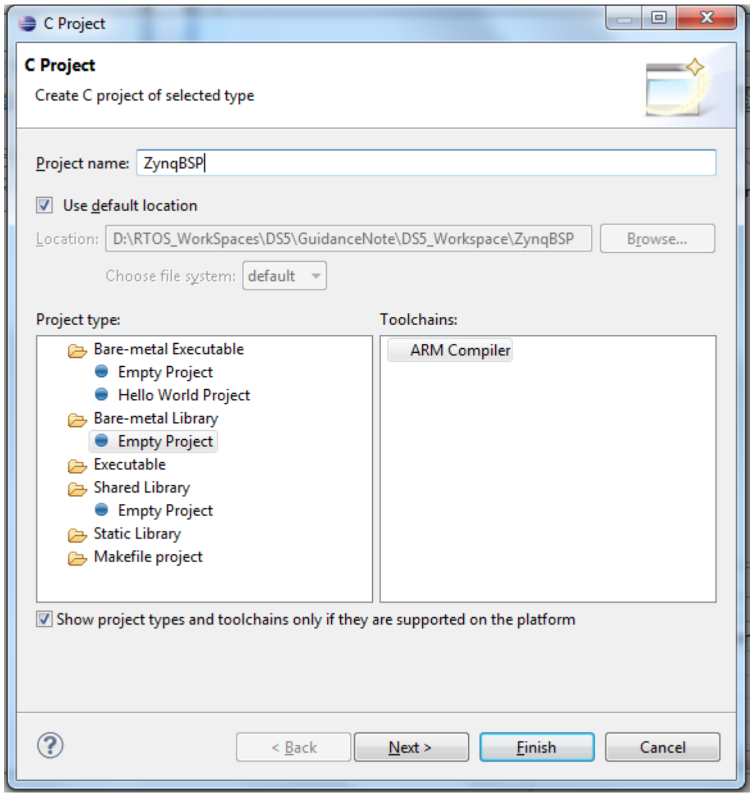
Figure 21: **Build Project Menu Option**

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

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

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

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**.
   b. Name the project (suggested name: `ZynqBSP`) and click **Next** (see the figure below).



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

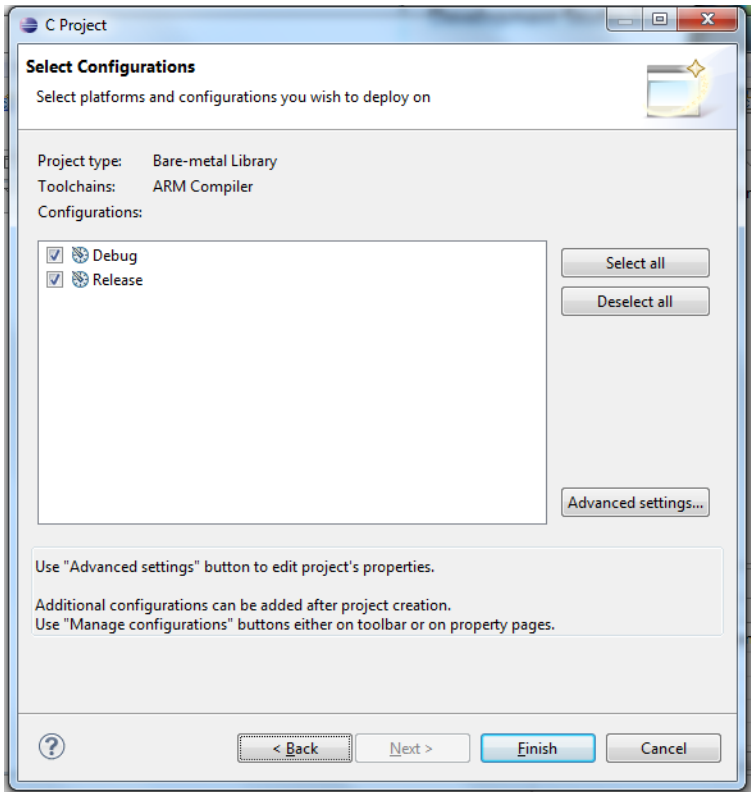
Project type:
- Bare-metal Executable
  - Empty Project
  - Hello World Project
- Bare-metal Library
  - Empty Project
- Executable
- Shared Library
  - Empty Project
- Static Library
- Makefile project

Toolchains:
- ARM Compiler

[✓] Show project types and toolchains only if they are supported on the platform

Figure 22: **Project Name and Type Selections Screen**

c. The Select Configurations screen appears (see the figure below). Verify that the information shown is correct and click **Finish**.



C Project

Select Configurations

Select platforms and configurations you wish to deploy on

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

- [✓] Debug
- [✓] Release

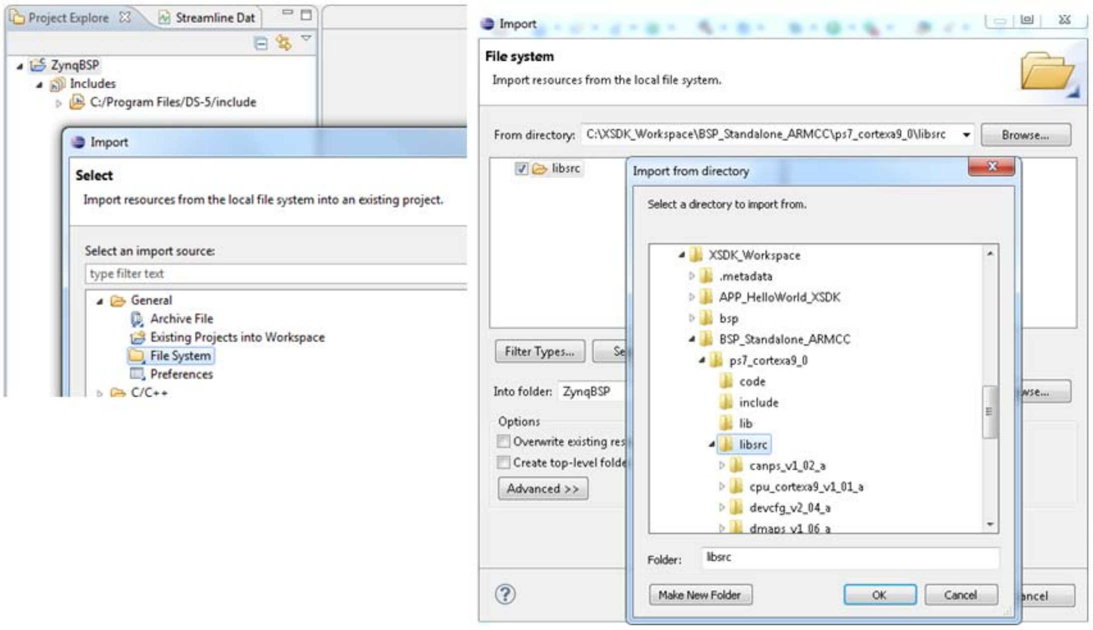
Use "Advanced settings" button to edit project's properties.

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

Figure 23: **"Select Configurations" Screen**

d. Right-click the Zynq device BSP library project and import the Xilinx SDK-generated (ARMCC) BSP source tree, specifically, the directory tree below `libsrc`, as shown in Figure 24, below.

e. Browse to the folder `BSP_Standalone_ARMCC\ps7_cortexa9_0\libsrc` and click **OK** (see the figure below).



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

From directory: C:\XSDK_Workspace\BSP_Standalone_ARMCC\ps7_cortexa9_0\libsrc

☑ libsrc

Into folder: ZynqBSP

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

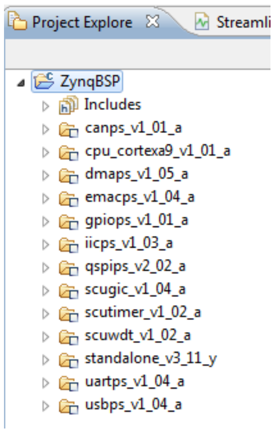
Folder: libsrc

Figure 24: **Importing the BSP Library Source**

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

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

The folders shown in the following illustration are now included in the ZynqBSP project, as shown below:



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

Figure 25: **BSP Project Folders**

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

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

b. In the Properties wizard, navigate to **C/C++ Build** > **Settings.** (See the figure below.)

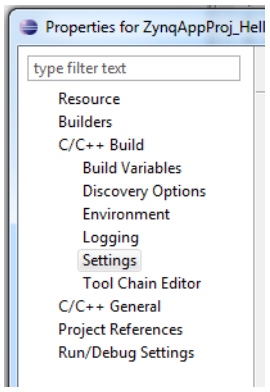
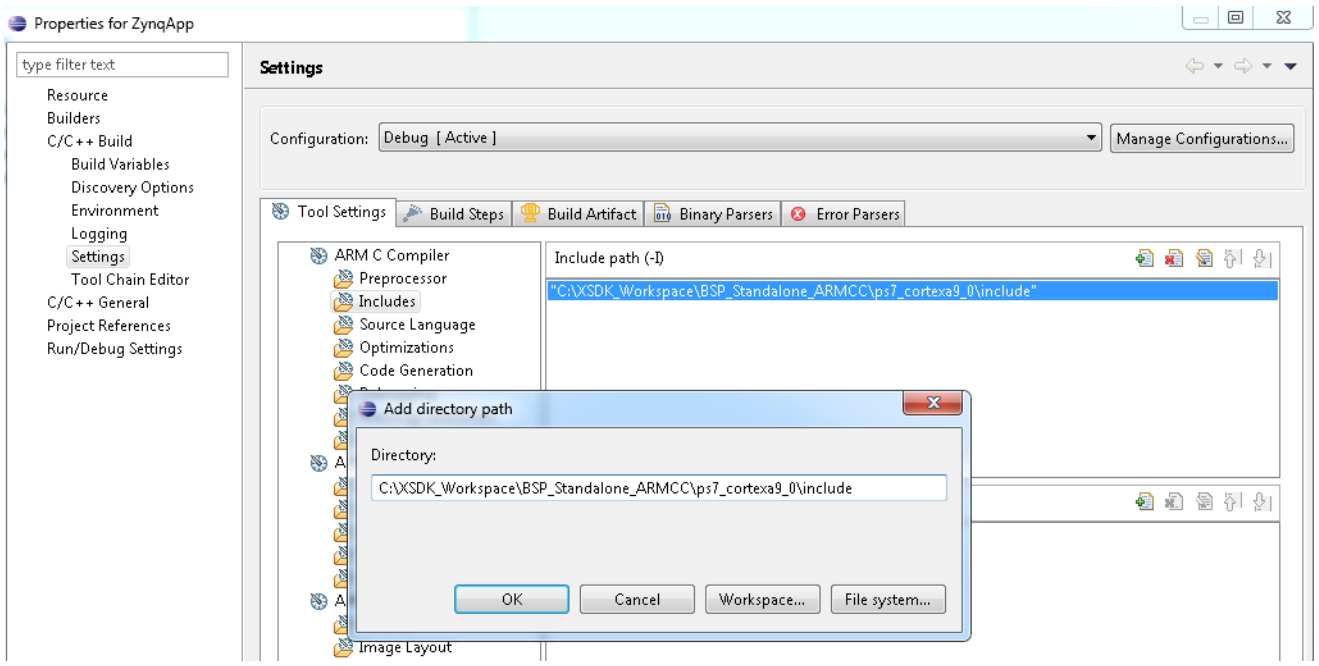


Figure 26: **Properties Wizard, C/C++ Build Options**

4. 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 as shown below (also see Figure 27):

```
-I <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"

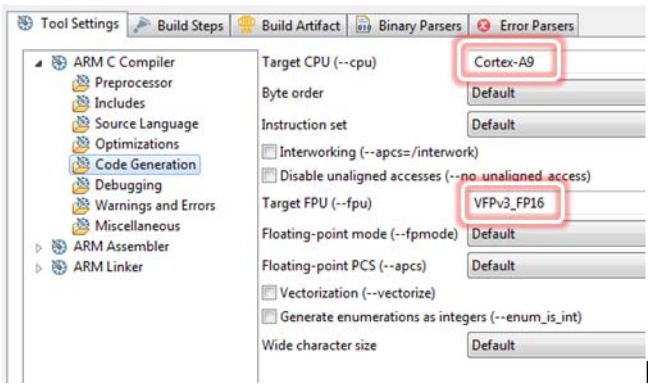
Directory: C:\XSDK_Workspace\BSP_Standalone_ARMCC\ps7_cortexa9_0\include

Figure 27: **ARM C Compiler Settings**

- In the **ARM C Compiler**, set the Code Generation parameters specific to the Zynq device:

  `--cpu Cortex-A9`

  `--fpu VFPv3_FP16`



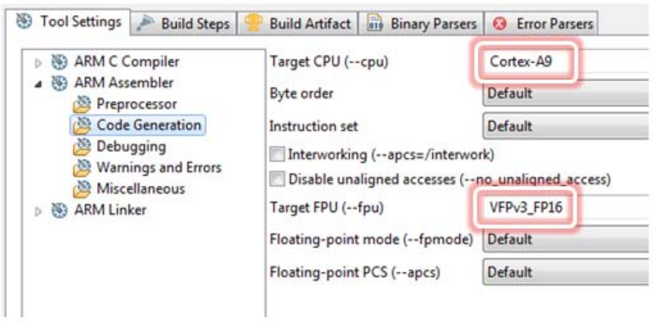
Target CPU (--cpu) Cortex-A9
Byte order Default
Instruction set Default
Interworking (--apcs=/interwork)
Disable unaligned accesses (--no_unaligned_access)
Target FPU (--fpu) VFPv3_FP16
Floating-point mode (--fpmode) Default
Floating-point PCS (--apcs) Default
Vectorization (--vectorize)
Generate enumerations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

Figure 28: ***ARM C Compiler*, Code Generation Parameter Settings**

- In the **ARM Assembler**, set the Code Generation parameters specific to the Zynq device:

  `--cpu Cortex-A9`

  `--fpu VFPv3_FP16`



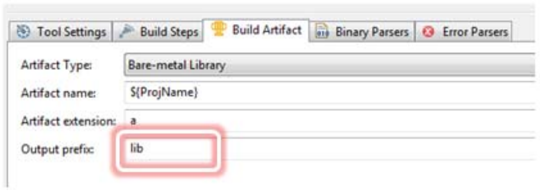
Target CPU (--cpu) Cortex-A9
Byte order Default
Instruction set Default
Interworking (--apcs=/interwork)
Disable unaligned accesses (--no_unaligned_access)
Target FPU (--fpu) VFPv3_FP16
Floating-point mode (--fpmode) Default
Floating-point PCS (--apcs) Default

Figure 29: ***ARM Assembler*, Code Generation Parameter Settings**

a. Click the **Build Artifacts** tab and make the following settings (also shown in Figure 30):

   Artifact extension: `a`

   Output Prefix: `lib`



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

Figure 30: **Build Artifacts Settings**

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

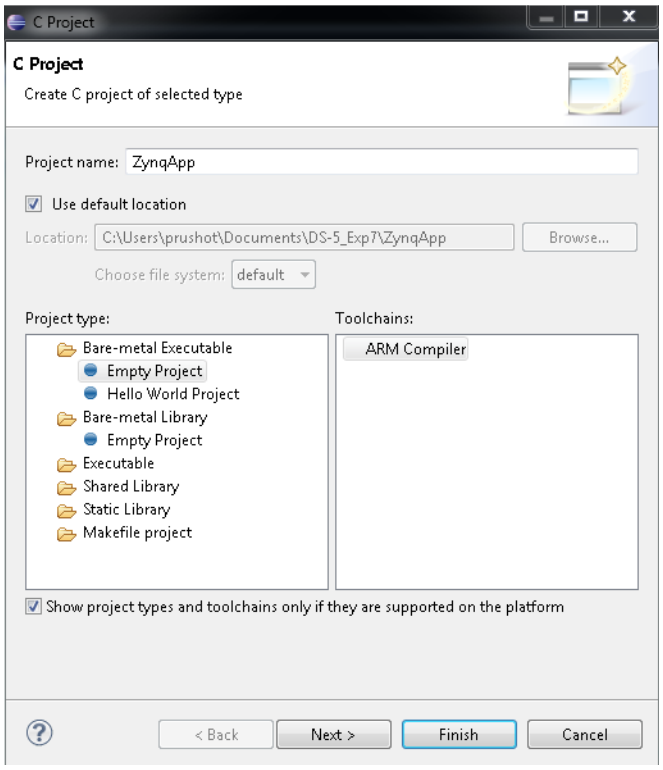
```
'Finished building target: libZynqBSP.a'
' '

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

Step 3: DS-5 Tools, Create a DS-5 Application Project for the Zynq Device

Use the following 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**.
   b. In the C Project screen (shown in the figure below), name the project (suggested name: `ZynqAPP`).
   c. Click **Next**.



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

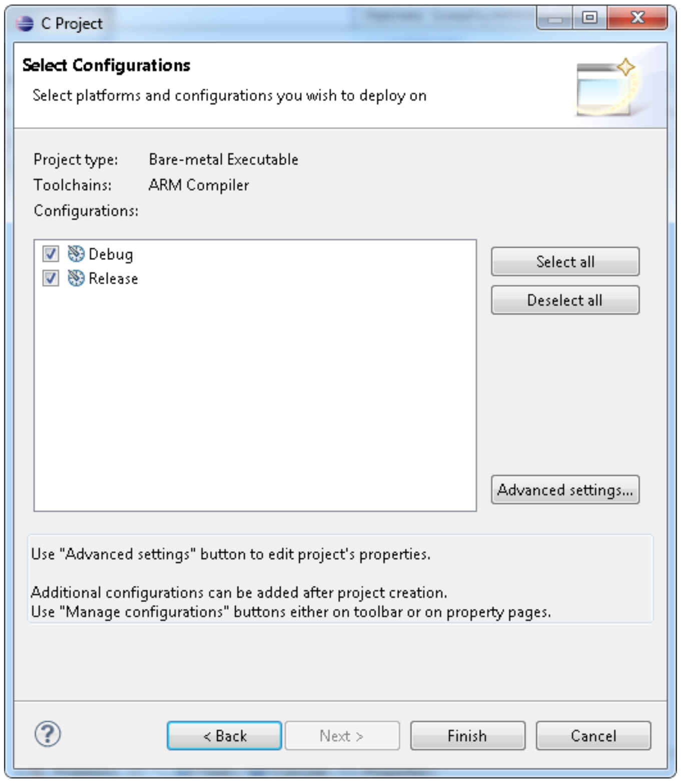
Project type:
- Bare-metal Executable
  - Empty Project
  - Hello World Project
- Bare-metal Library
  - Empty Project
- Executable
- Shared Library
- Static Library
- Makefile project

Toolchains:
- ARM Compiler

[✓] Show project types and toolchains only if they are supported on the platform

Figure 31: **C Project Screen**

d. A Select Configurations screen appears. Verify that the selections are correct (see the figure below) and click **Finish** to close the C Project wizard.



Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Bare-metal Executable

Toolchains: ARM Compiler

Configurations:

[✓] Debug

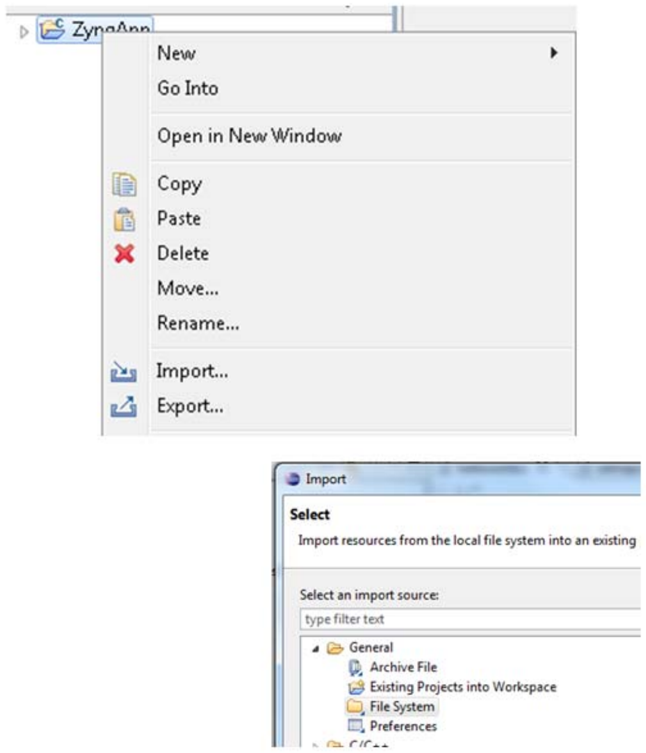
[✓] Release

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.

Figure 32: **“Select Configurations” Screen**

e. Right-click the ZynqAPP application project, select **Import** (see the figure below), and navigate to the `Hello World` application project (`App_HelloWorld_XSDK`) created in Step 1: Xilinx SDK, Create the Standalone Board Support Package.

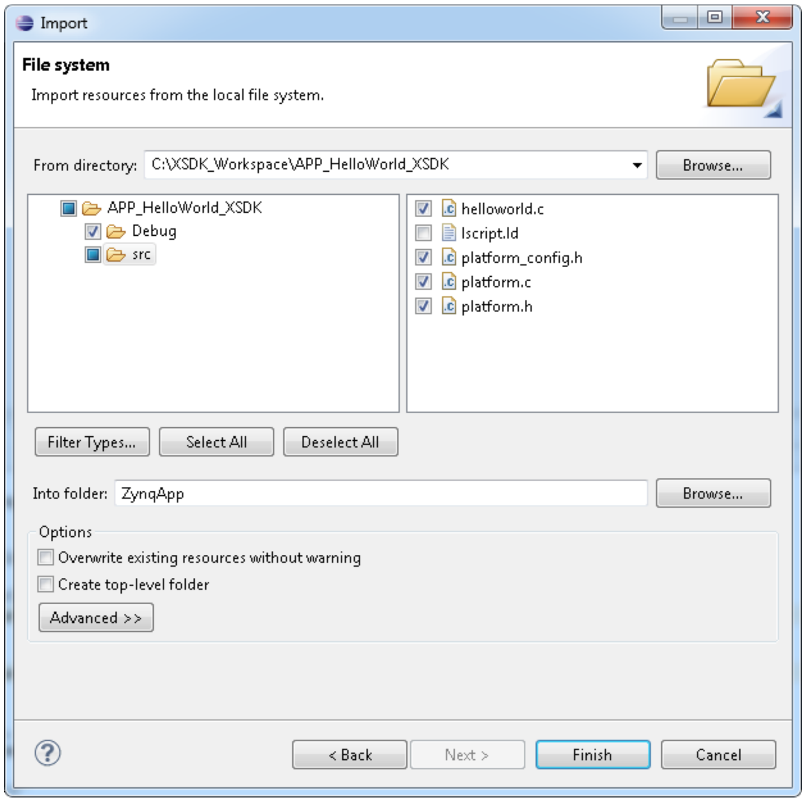


Import resources from the local file system into an existing

Figure 33: **Import Menu Command and Import "Select" Screen**

f. In the Import screen (shown in the figure below), include all the C Source and Header files (`*.c, *.h, *.S`). Also ensure that the `helloworld.c` check box is selected.

g. Click **Finish**.



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

Figure 34: **Import "File system" Screen**

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

   a. Right-click the ZynqApp project 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:

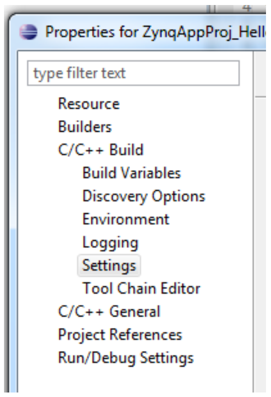
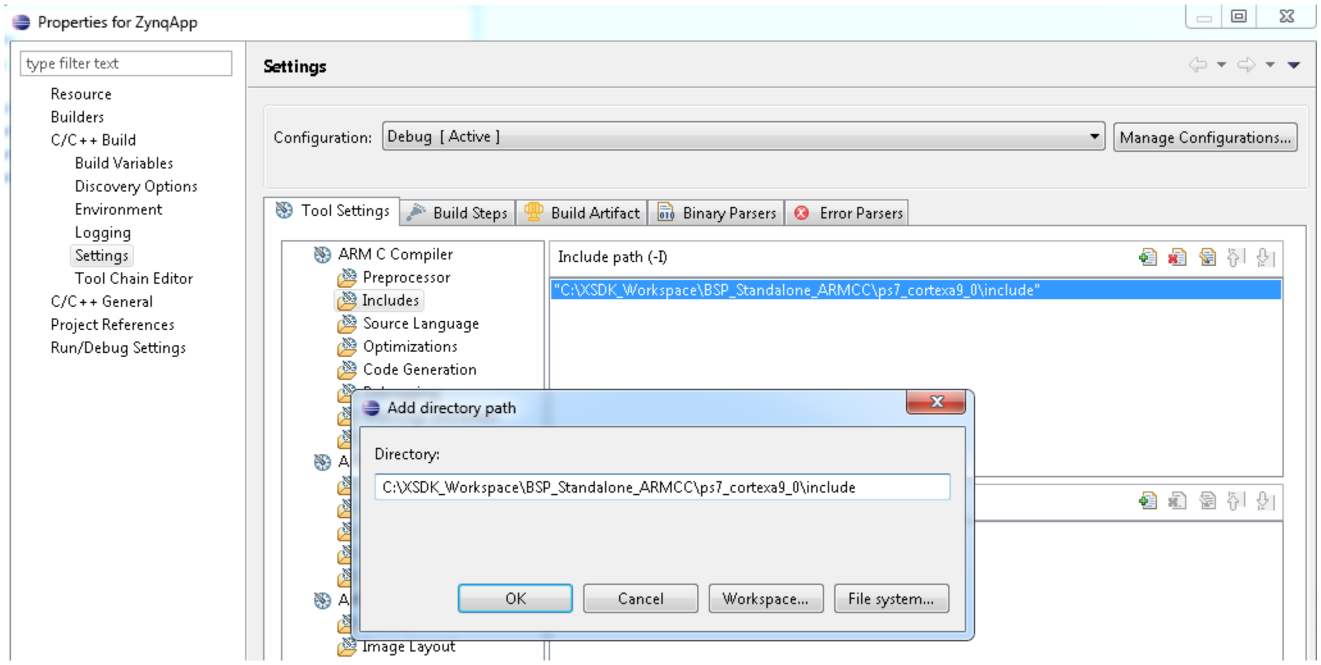


Figure 35: **ZynqApp Properties Selections**

- In the **ARM C Compiler**, set the `Include` paths as follows (and as shown in the figure below) 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
```



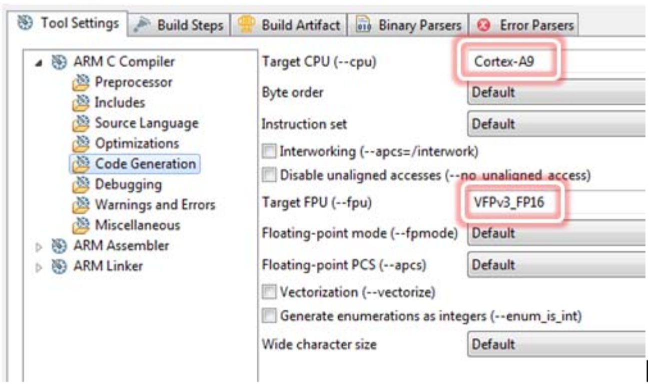
"C:\XSDK_Workspace\BSP_Standalone_ARMCC\ps7_cortexa9_0\include"

C:\XSDK_Workspace\BSP_Standalone_ARMCC\ps7_cortexa9_0\include

Figure 36: **ZynqApp Properties, `include` Paths**

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

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

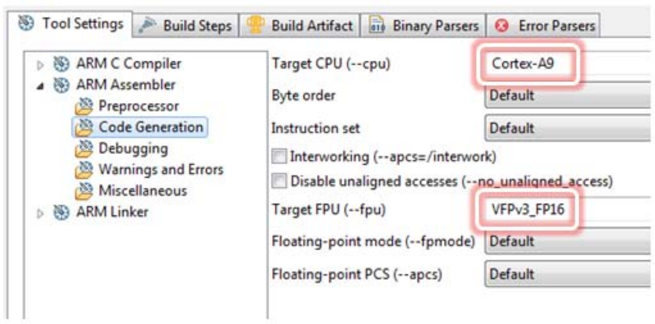


Target CPU (--cpu) Cortex-A9
Byte order Default
Instruction set Default
Interworking (--apcs=/interwork)
Disable unaligned accesses (--no_unaligned_access)
Target FPU (--fpu) VFPv3_FP16
Floating-point mode (--fpmode) Default
Floating-point PCS (--apcs) Default
Vectorization (--vectorize)
Generate enumerations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

Figure 37: ***ARM C Compiler* Code Generation Parameters**

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

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


Tool Settings | Build Steps | Build Artifact | Binary Parsers | Error Parsers

ARM C Compiler
ARM Assembler
- Preprocessor
- Code Generation
- Debugging
- Warnings and Errors
- Miscellaneous

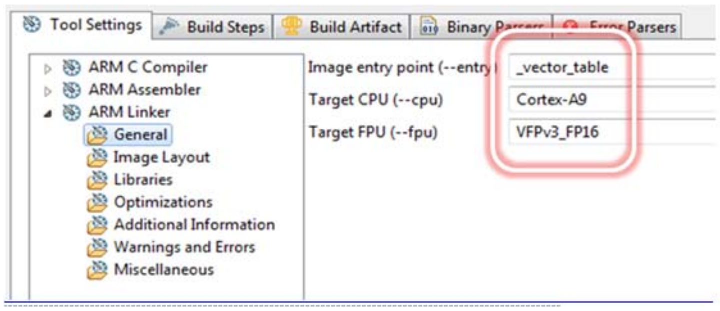
ARM Linker

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

Figure 38: ***ARM Assembler*, Code Generation Parameters**

d. In the **ARM Linker**, set the General parameters specific to the Zynq architecture as follows (also shown in the figure below):

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


Tool Settings | Build Steps | Build Artifact | Binary Parsers | Error Parsers

ARM C Compiler
ARM Assembler
ARM Linker
- General
- Image Layout
- Libraries
- Optimizations
- Additional Information
- Warnings and Errors
- Miscellaneous

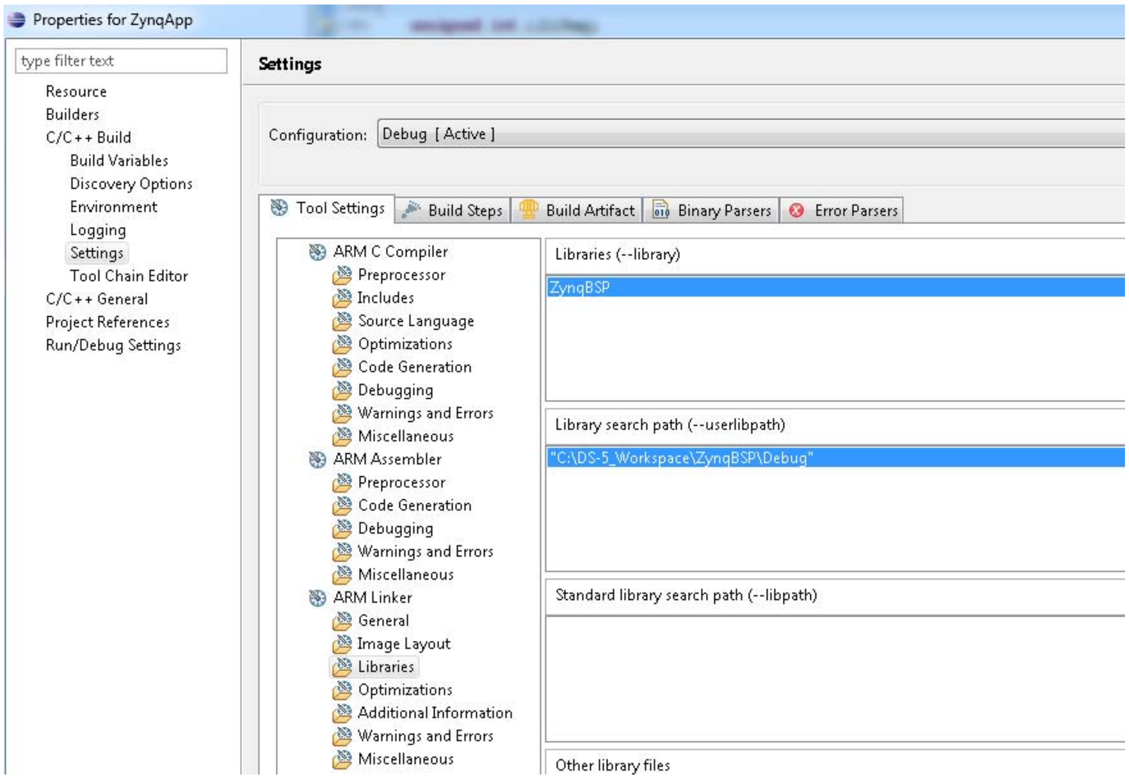
Image entry point (--entry): _vector_table
Target CPU (--cpu): Cortex-A9
Target FPU (--fpu): VFPv3_FP16

Figure 39: ***ARM Linker*, General Parameters**

e. In the **ARM Linker > Libraries**, add the reference BSP build created in Step 2: DS-5 Tools, Import and Build the BSP:

Libraries (`--library`): `ZynqBSP`

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



Configuration: Debug [Active]

Libraries (--library)

ZynqBSP

Library search path (--userlibpath)

"C:\DS-5_Workspace\ZynqBSP\Debug"

Standard library search path (--libpath)

Other library files

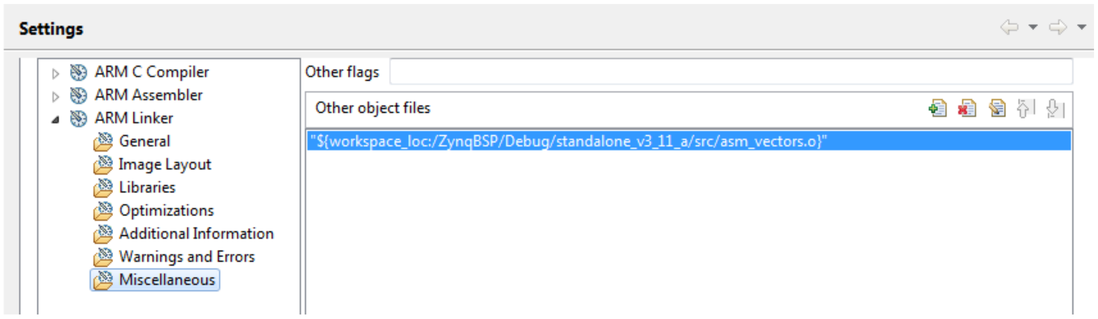
Figure 40: **ZynqApp Settings: Adding the Reference BSP to the ARM Linker Libraries**

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`"

h. Select the `asm_vectors.o` file (shown in the figure below).



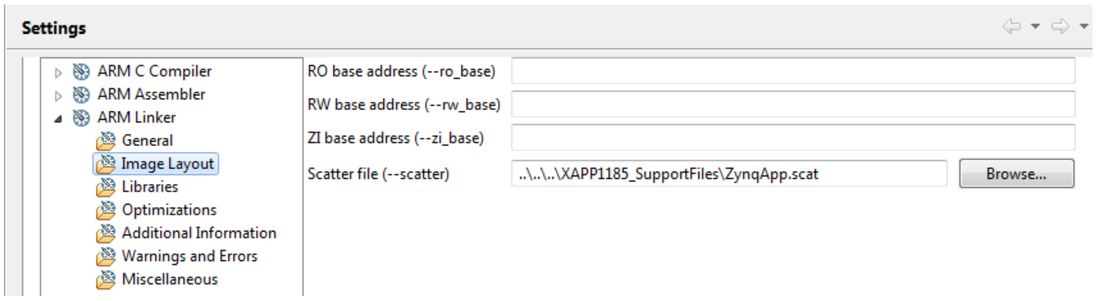
Other flags

Other object files

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

Figure 41: **"Other object files" Selection**

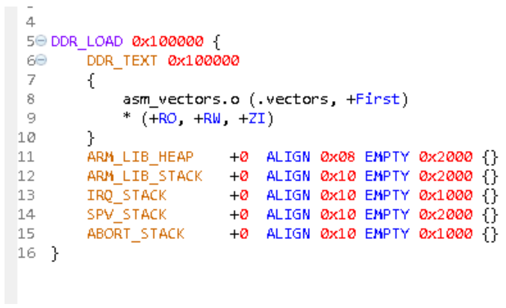
i. In the ARM Linker, select **Image Layout** (shown in the figure below) and define the Linker Script, described here as a "scatter" file.



RO base address (--ro_base)
RW base address (--rw_base)
ZI base address (--zi_base)
Scatter file (--scatter) ..\..\..\XAPP1185_SupportFiles\ZynqApp.scat

Figure 42: **ARM Linker, Image Layout Settings**

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

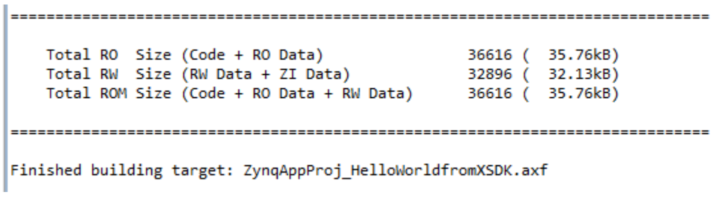


```

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 {}
}
```

Figure 43: **Sample Scatter File Image Layout**

k. Compile the application. As a reference, the following figure 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
```

Figure 44: **Compiled Application File Size**

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.
05/06/14	2.0	Steps have been added for using a batch script to automate project generation steps. The original, manual procedure is now the Appendix section of this document.

Notice of Disclaimer