

ARM DS-5 Tools and Avnet ZED Series

#9

SDK to DS-5 Migration



September 2013
Version 02

Table of Contents

ARM DS-5 Tools and ZedBoard Series	3
Required Installations	4
Technical Support	5
Xilinx SDK and ARM DS-5	6
Migration Overview	7
Boot the ZED Target from Xilinx SDK	7
Create a Standalone ARM DS-5 Project	13
Boot the ZED Target using the ARM DS-5 Executable	19
Revision History	23
Resources	23

ARM DS-5 Tools and ZedBoard Series

This tutorial is one in a series of step by step instruction manuals. Together they document the procedures necessary to utilize the ARM Development Studio 5 (DS-5™) Software Suite and the DSTREAM Debugging tools with the Avnet Zynq Evaluation and Development (ZED) boards. These tutorials can be used on their own, or in combination with Avnet online videos and OnRamp Technical Session™.

The ARM software and hardware tools provide a powerful debugging suite for processor-based systems built around the dual Cortex-A9 cores present in the Xilinx Zynq SoC, at the heart of the Avnet ZED boards. A Linux software developer can simultaneously debug applications and kernel module code, with separate control over each thread. You can step through Linux boot code, first stage bare metal boot code, and bare metal applications. When used in concert with the Xilinx Vivado tools for FPGA fabric development, the ARM debugger and Internal Logic Analyzer (ILA) IP can be cross-triggered to stop on software and hardware breakpoints, or when a hardware event occurs. For difficult-to-isolate intermittent faults, DS-5 provides access to the Cortex-A9 on-chip Trace facility. Once your embedded system is running correctly, DS-5 uses Streamline, a graphical system profiler, to identify performance bottlenecks in your design to ensure top-shelf operation.

This tutorial series begins with the most basic tool configuration and board connection. It takes you all the way through to the most complex aspects of hardware/software co-debugging to root out design errors that are otherwise apparent only in very complex use cases, or worse, after a product is released. Together the ARM DS-5 tools, Xilinx Vivado and Avnet ZED boards provide an unparalleled combination to compress design timelines, cut project costs and optimize your product for the marketplace.

Required Installations

Software

The recommended software for this tutorial series is:

- ARM Development Studio 5 (Exact version used is 5.14, build 1702)
- Xilinx ISE WebPACK 14.5 (Free license and download from Xilinx website)
- Cypress CY7C64225 USB-to-UART Bridge Driver (for ZedBoard serial output)
- Silicon Labs CP2104 USB-to-UART Bridge Driver (for MicroZed serial output)
- Tera Term (Exact version used is V4.75)
- Xilinx Software Development Kit, version 14.5
- For hardware/software co-debugging, Xilinx Vivado 2013.2

Hardware

The targeted hardware consists of the following:

- PC workstation with at least 5 GB RAM, 30GB free hard disk space, Windows 7 64-bit operating system, and a wired GB Ethernet connection
- Available SD card slot on PC or external USB-based SD card reader
- One of:
 - Avnet ZedBoard Kit (**AES-Z7EV-7Z020-G**)
 - USB cable (Type A to Micro-USB Type B)
 - 4GB SD card
 - 12v Power supply
 - Avnet MicroZed Kit (**AES-Z7MB-7Z010-G**)
 - USB cable (Type A to Micro-USB Type B)
 - 4GB SD card
- Avnet ZedBoard Debug Adapter Kit (**AES-ZBDB-ADPT-G**)
 - 14-pin Xilinx PC4 ribbon cable
- ARM DSTREAM unit and Keil pod with wide cable connector
 - 20-pin JTAG ribbon cable
 - USB cable (Type A to Printer)
 - 5v Power supply
- CAT-5 Ethernet cable

Technical Support

For technical support with any of the instructions, please contact your local Avnet/Silica FAE or visit the support forums:

<http://www.zedboard.org/forum>

<http://www.microzed.org/forum>

Additional technical support resources are listed below.

ZedBoard Kit/MicroZed Kit support page with Documentation and Reference Designs

<http://www.zedboard.org/content/support>

<http://www.microzed.org/content/support>

For Xilinx technical support, you may contact your local Avnet/Silica FAE or Xilinx Online Technical Support at www.support.xilinx.com . On this site you will also find the following resources for assistance:

- Software, IP, and Documentation Updates
- Access to Technical Support Web Tools
- Searchable Answer Database with Over 4,000 Solutions
- User Forums
- Training - Select instructor-led classes and recorded e-learning options

Contact your Avnet/Silica FAE or Avnet Support for any additional questions regarding the reference designs, kit hardware, or if you are interested in designing any of the kit devices into your next design.

<http://www.em.avnet.com/techsupport>

For ARM technical support, you may contact your local Avnet/Silica FAE or ARM Online Technical Support at www.arm.com/support .

Xilinx SDK and ARM DS-5

If you have been developing processor-based embedded systems for the Xilinx Zynq processor, then in all likelihood you have been using the Xilinx Software Development Kit to perform the initial bring-up of the hardware. The main advantage in using the SDK at this point is to take advantage of the embedded environment created automatically by the SDK tool suite, including:

1. First stage boot loader
2. Board Support Package
3. Software drivers for peripherals in the Zynq processor system
4. Software drivers for Programmable Logic peripherals from the Xilinx IP Library
5. Basic test applications for Hello World, Memory and Peripheral Tests

Once you have a basic system running, you may wish to use the ARM DS-5 tool suite to take advantage of the advanced debugging facilities and DSTREAM hardware. To do this you will need to migrate your SDK project to the DS-5 tools.

Creation of a seamless method to export the underlying software infrastructure from an SDK project to an external tool suite such as DS-5 is a project that is on the Xilinx roadmap, but it has not yet come to fruition. In the interim, this document provides the framework for a work-around procedure that uses the SDK code for bare metal processor system initialization and first-stage boot loading. Using this method you can develop your own application code within DS-5, and integrate it with the SDK-generated initialization code to allow you boot the ZED target and run your custom applications.

You will need application source files and SD card files to complete these instructions. The files are part of the download package that included this tutorial. Browse to the location on your host where the download package was decompressed, and look for the following folder:

<Download package folder>\Support09

For the purposes of this tutorial we will assume the files exist in folder:

C:\OnRamps\Support09

Migration Overview

You can create a processor-based hardware platform for the ZED target using one of the Xilinx Hardware tools suites (PlanAhead or Vivado). Once you have the processor system, export the hardware platform to an SDK project. Details for these procedures can be found in numerous Avnet OnRamps and Speedways, as well as on the Xilinx Support site.

The goal of this tutorial is to create a boot package for the ZED target using the Xilinx SDK, and to port elements of that package to the ARM DS-5 tool suite. We start with an SDK project based on a simple hardware system, and demonstrate how to integrate a portion of the support libraries within a DS-5 software project. The resulting executable from DS-5 can then be combined with the boot image from SDK to allow the ZED target to boot successfully from an SD card.

The procedure can be broken down into 5 steps:

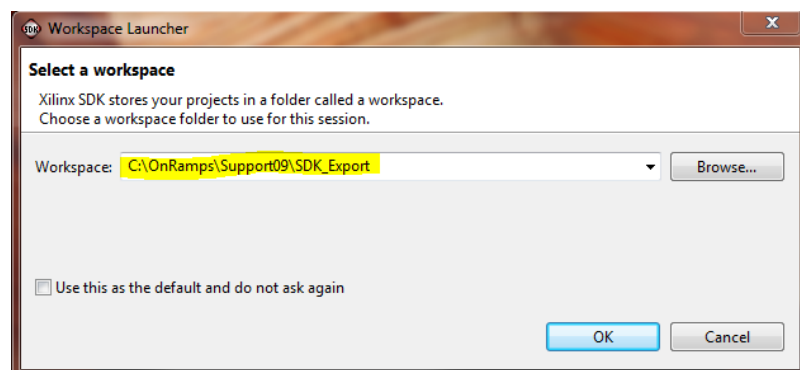
1. Create a boot package to validate the processor system within the SDK.
2. Extract
3. And whatever else there is.

For this tutorial, we will begin with an SDK 2013.2 workspace exported from PlanAhead 14.6. You will find the workspace at:

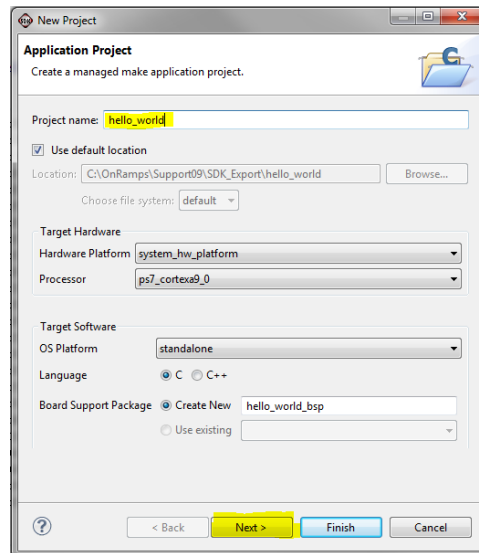
C:\OnRamps\Support09\SDK_Export

Boot the ZED Target from Xilinx SDK

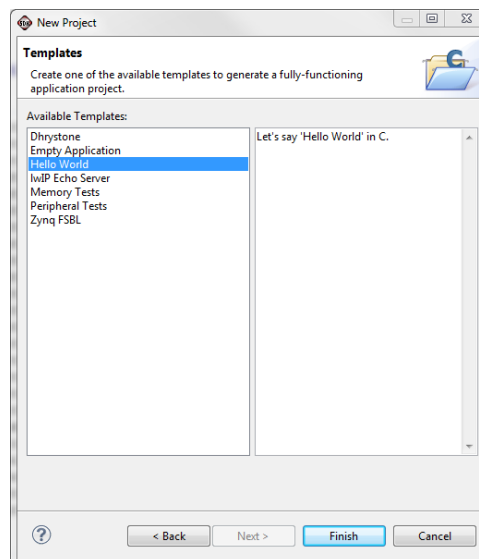
1. Start the SDK and select the supplied workspace. You can copy the workspace to any location on your host (no spaces in the pathname), but for the purposes of these instructions we will assume it remains at its default location.



2. Create a Hello World application that will send output to the UART.
 - a. From the main menu, select **File | New | Application Project**.
 - b. Name the project **hello_world** and click the **Next** button. The project will target processor 0 in the Zynq chip by default, and a separate board support package **hello_world_bsp** will be generated, based on the supplied hardware platform.



- c. Select the **Hello World** template to generate C code for our application, and click the **Finish** button. The projects will build after creation.



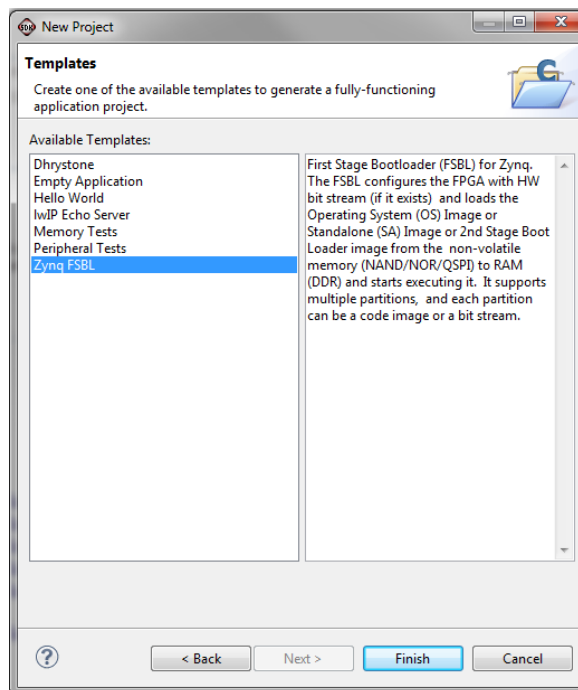
3. Modify the **helloworld.c** main function to print “**Hello World from SDK application**” to differentiate this from a DS-5 program.

```
int main()
{
    init_platform();

    print("Hello World from SDK application\n\r");

    return 0;
}
```

4. In order to boot from an SD card, independent of any JTAG connection, we need to have a boot loader. The SDK can easily create one for our ZED target.
 - a. From the main menu, select **File | New | Application Project**.
 - b. Name the project **zynq_fsbl_0** and click the **Next** button.
 - c. Select the **Zynq FSBL** template and click the **Finish** button to create the boot loader and its associated board support project (zynq_fsbl_0_bsp). As before, the projects will build after creation.



5. The FSBL performs a number of initialization steps and hands off control to the next application in the boot sequence. In this case, that application will be the hello world program created earlier. To reinforce the handoff, we can make a simple modification to the FSBL code to print a message to the UART, just prior to the transfer of control to our application.
 - a. Under the **zynq_fsbl_0** project, expand the **src** entry and double-click **fsbl_debug.h** to open the file in the source editor. Add the highlighted lines shown below and save the file.

```

*****

#ifdef _FSBL_DEBUG_H
#define _FSBL_DEBUG_H

#ifdef __cplusplus
extern "C" {
#endif

// Remove this line to turn off the very specific debug messages.
#define FSBL_DEBUG_SPECIFIC 1

#define DEBUG_GENERAL 0x00000001 /* general debug messages */
#define DEBUG_INFO 0x00000002 /* More debug information */
#define DEBUG_SPECIFIC 0x00000004 /* Very specific debug information */

#if defined (FSBL_DEBUG_INFO)
#define fsbl_dbg_current_types ((DEBUG_INFO) | (DEBUG_GENERAL))
#elif defined (FSBL_DEBUG)
#define fsbl_dbg_current_types (DEBUG_GENERAL)
#elif defined (FSBL_DEBUG_SPECIFIC)
#define fsbl_dbg_current_types (DEBUG_SPECIFIC)
#else
#define fsbl_dbg_current_types 0
#endif

```

- b. Double-click **main.c** to open the file in the source editor. Add the highlighted lines shown below and save the file to rebuild the project. Load boot image is found near line 510.

```

/*
 * Load boot image
 */
HandoffAddress = LoadBootImage();

fsbl_printf(DEBUG_INFO, "Handoff Address: 0x%08x\r\n", HandoffAddress);

/*
 * For Performance measurement
 */
#ifdef FSBL_PERF
XTime tEnd = 0;
fsbl_printf(DEBUG_GENERAL, "Total Execution time is ");
FsblMeasurePerfTime(tCur, tEnd);
#endif

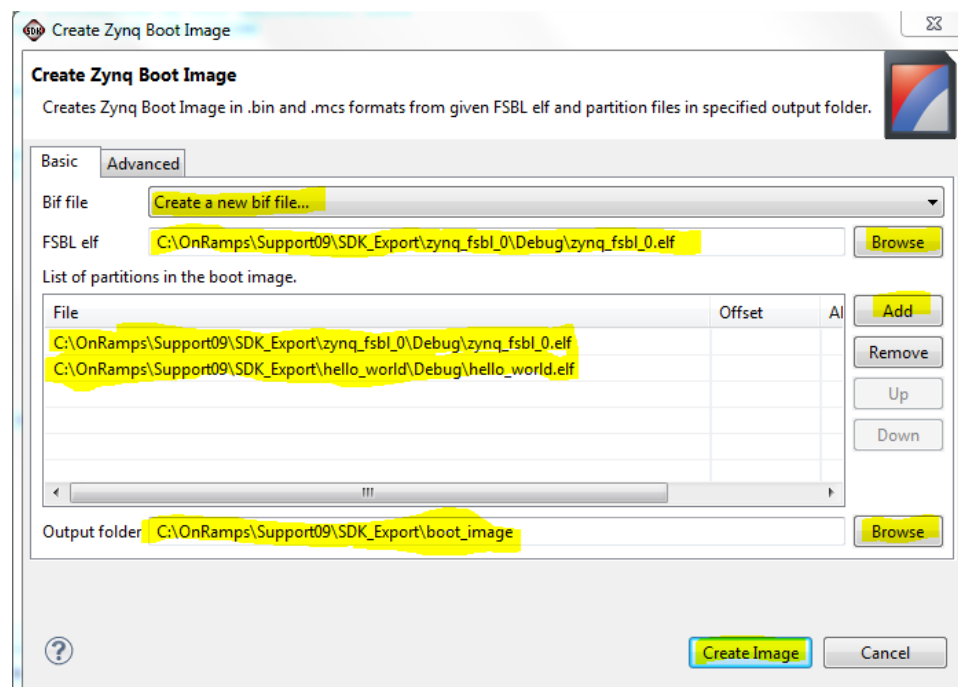
fsbl_printf(DEBUG_SPECIFIC, "FSBL Handoff to address 0x%08X is about to occur\r\n", HandoffAddress);
fsbl_printf(DEBUG_SPECIFIC, "\r\n");

/*
 * FSBL handoff to valid handoff address or
 * exit in JTAG
 */
FsblHandoff(HandoffAddress);

```

6. Next, we will create a boot image that we will copy to the SD card of our ZED target to test our processor system.

- a. From the main menu, select **Xilinx Tools | Create Zynq Boot Image**.
- b. Set the parameters for our boot file in the pop-up dialog.
 - i. In the **Bif file** field, select **Create a new bif file**.
 - ii. In the FSBL elf file field, click **Browse** to and locate the **zynq_fsbl_0.elf** file in the Debug folder of your zynq_fsbl_0 project.
 - iii. Click the **Add** button and browse to the **hello_world.elf** file in the Debug folder of your **hello_world** project.
 - iv. In the Output folder field, click the **Browse** button to specify the **boot_image** folder under your SDK workspace.



- v. Click the **Create Image** button to generate the **hello_world.bin** file that will boot our system.

7. In the **boot_image** folder, rename **hello_world.bin** to **boot.bin** and copy it to the SD card. The ZED target will only recognize a file named boot.bin at load time. Insert the SD card on the underside of your ZED target.

Set the ZED target Boot Mode to SD boot as shown below:

For ZedBoard:

To begin the procedure, set the ZedBoard Boot Mode to SD boot using jumpers JP11 to JP7 set to the following:

	JP11	JP10	JP9	JP8	JP7
Position	SIG-GND	3V3-SIG	3V3-SIG	SIG-GND	SIG-GND

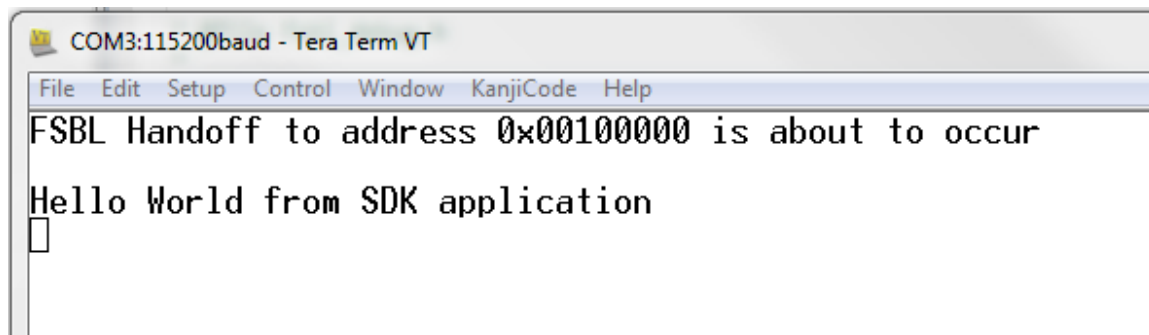
For MicroZed:

To begin the procedure, set the MicroZed Boot Mode to JTAG only using jumpers JP3 to JP1 set to the following:

	JP3	JP2	JP1
Position	2-3	2-3	1-2

You should have the USB-UART of your ZED target connected to your host PC.

Power the ZED target to validate that the hardware platform and application work correctly. You can monitor the output in Tera Term Pro. Note that the FSBL modification informs us that the handoff address is 0x100000, which is where our application code must be loaded in order to run correctly.

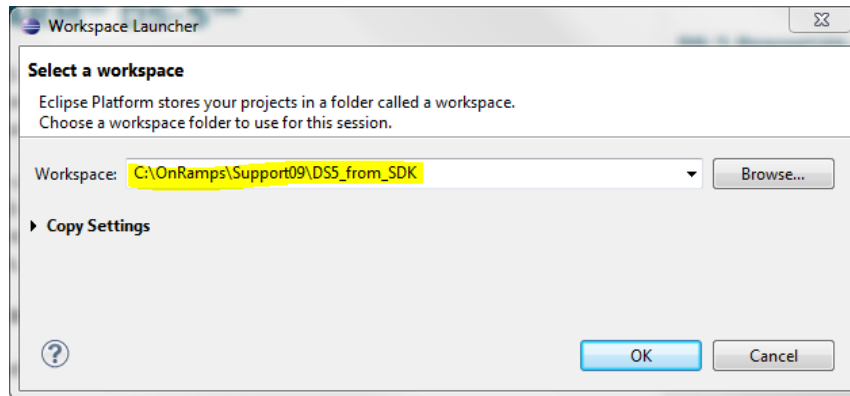


The screenshot shows a Tera Term VT terminal window titled 'COM3:115200baud - Tera Term VT'. The menu bar includes File, Edit, Setup, Control, Window, KanjiCode, and Help. The terminal output displays two lines: 'FSBL Handoff to address 0x00100000 is about to occur' followed by 'Hello World from SDK application' on the next line. A small cursor is visible at the end of the second line.

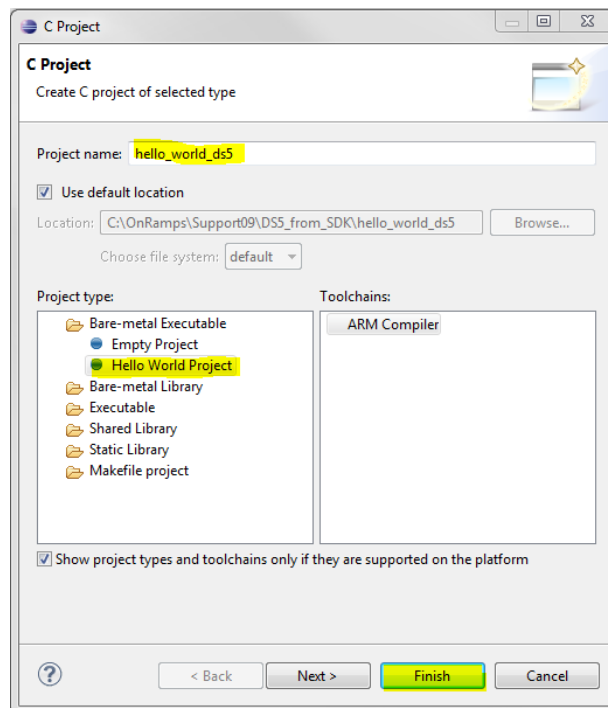
Create a Standalone ARM DS-5 Project

1. Open a DS-5 Workspace and create a Hello World software project. For the purposes of this tutorial, we will assume it is a new workspace in:

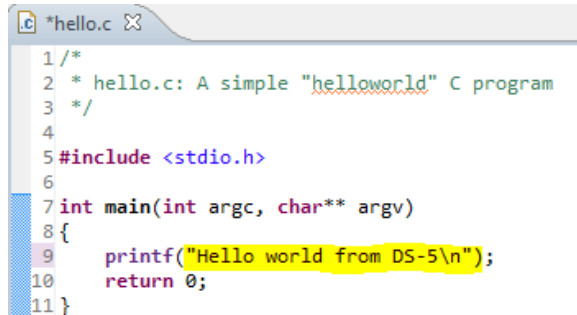
C:\OnRamps\Support09\DS5_from_SDK



- a. From the main menu, select **File | New | C Project**.
- b. Set the project name to **hello_world_ds5**. Set the Project type to **Hello World Project**. Click the **Finish** button to create the project.



- c. In the source window, update the printf line to read “Hello world from DS-5\n” to differentiate it from the SDK application. Save the changes. (File | Save)



```
1 /*
2 * hello.c: A simple "helloworld" C program
3 */
4
5 #include <stdio.h>
6
7 int main(int argc, char** argv)
8 {
9     printf("Hello world from DS-5\n");
10     return 0;
11 }
```

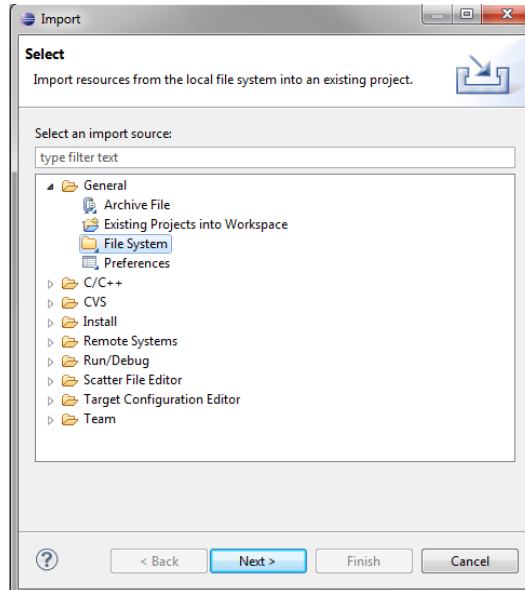
2. At this point we have an application that will build without error in DS-5, but it will not execute correctly on our ZED target because it lacks the infrastructure created by the SDK tool suite. So some additional changes must be made to our DS-5 project to bridge to the Zynq environment required to write to the UART (created by the SDK in the Zynq BSP that was integrated into the boot.bin file).

- a. **retarget.c** – The following C library functions make use of DS-5 semihosting to read or write characters to the debugger console: **fputc()**, **fgetc()**, and **_ttywrch()**. They must be retargeted to write to the UART as configured by the SDK for the Zynq platform. The scanf() function also requires that **__backspace()** be retargeted as well. In addition, to allow exceptions and signals to be handled correctly, the **ferror()** and **_sys_exit()** functions must be retargeted. The retarget.c file provides the changes to these functions to allow for operation on the Zynq platform.

Once you have the retarget.c file added to your DS-5 project, you should open it in the source window and examine the C code for complete descriptions of the implementation for each function.

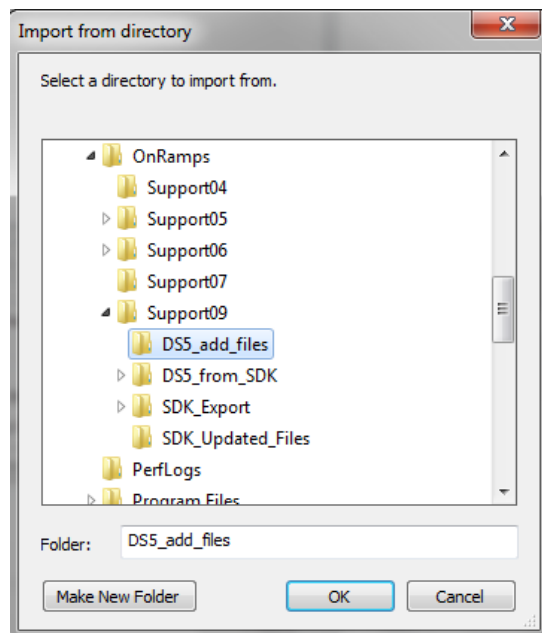
- i. From the main menu, select **File | Import**.

- ii. Select **General | File System** and click the **Next** button.

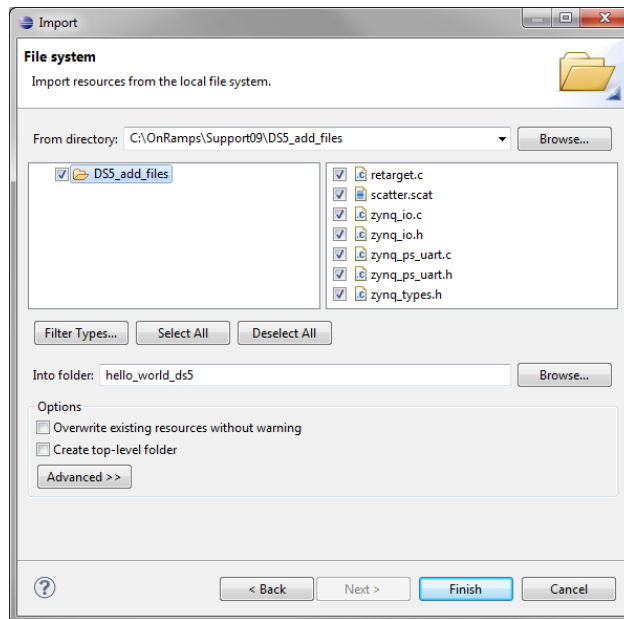


- iii. In the From directory box, click the Browse button and locate the **DS5_add_files** folder. If you used the recommended file path for installation, this will be located at:

C:\OnRamps\Support09\DS5_add_files

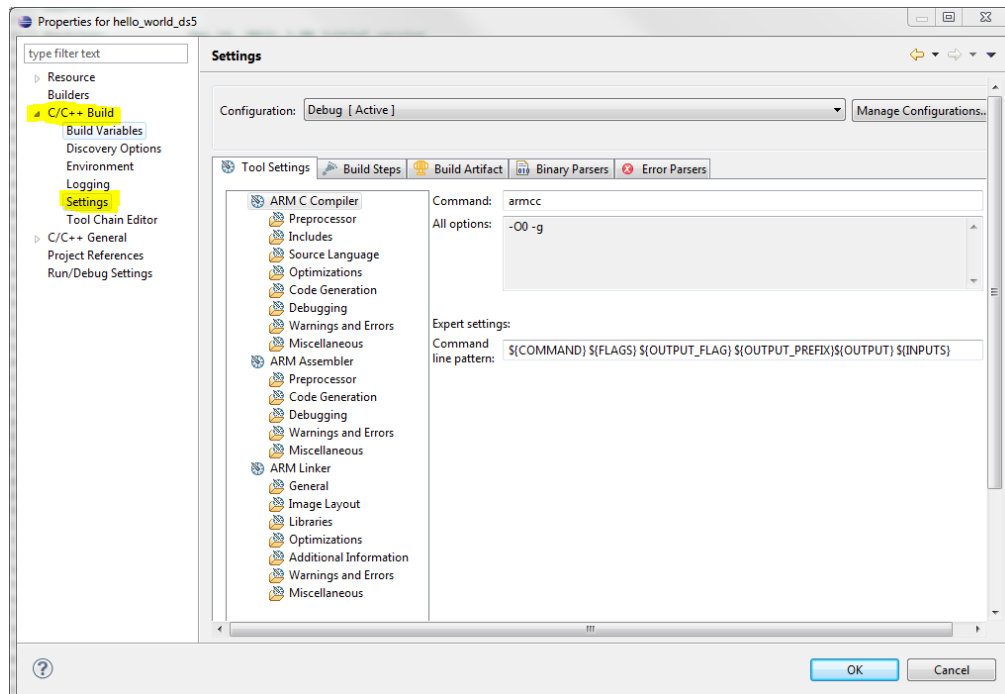


- iv. All the files in this folder are required, so **click the checkbox** next to **DS5_add_files** to select them all. Click the **Finish** button to add the files to the project.



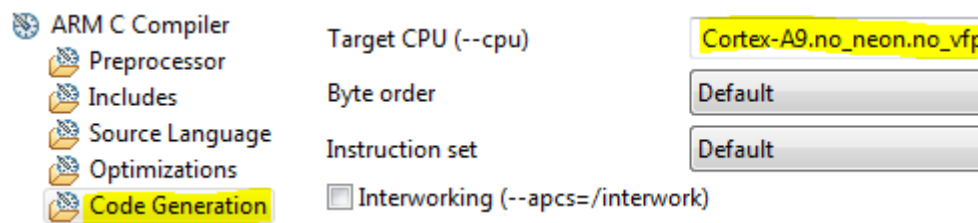
- b. **zynq_ps_uart.c** and **zynq_ps_uart.h** – these functions implement the Processor System UART read/write functions used by **retarget.c**.
- c. **zynq_io.c** and **zynq_io.h** – these functions implement the low level I/O routines called from the **zynq_ps_uart** files.
- d. **zynq_types.h** – this file redefines the standard data types to use the same names as in the Xilinx BSP.
- e. **scatter.scad** – the linker directive file that places the code and data in the proper memory locations. The main program is mapped to 0x100000, which is the location where the FSBL transfers control.

3. Now we have all the source code in place, there are still some final properties to set in the DS-5 build environment. Right-click on the `hello_world_DS5` project and select **Properties**. Expand **C/C++ Build** and select **Settings**.



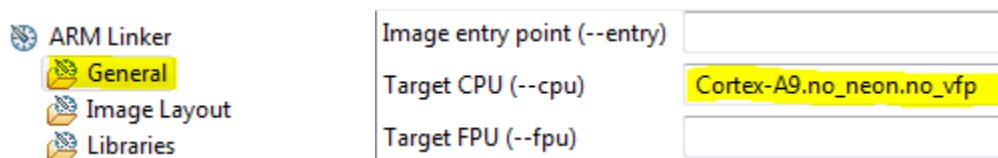
- i. Select **ARM C Compiler | Code Generation**. In the **Target CPU** field, specify no neon co-processor and no floating point with:

Cortex-A9.no_neon.no_vfp



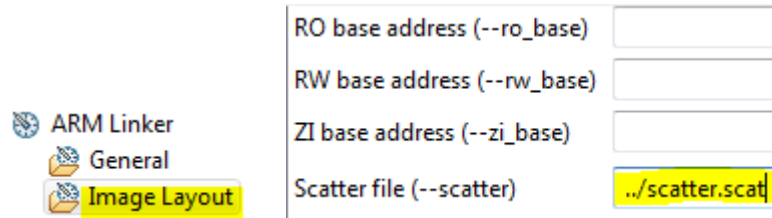
- ii. Select **ARM Linker | General**. In the **Target CPU** field, enter:

Cortex-A9.no_neon.no_vfp



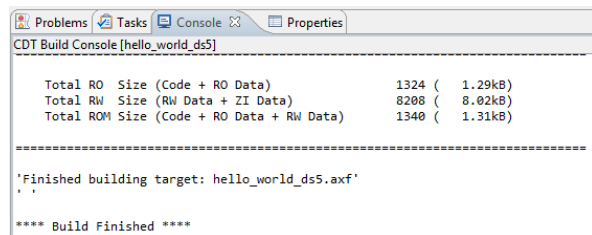
- iii. Select **ARM Linker | Image Layout**. In the **Scatter file** field, set the program memory map by pointing to the linker file. The relative path is in relation to the linker output folder Debug.

../scatter.scat



- b. Click **OK** to save the Project Properties.

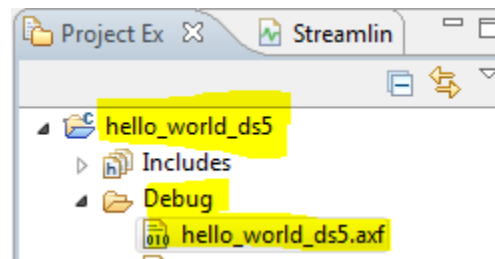
4. Generate the executable (hello_world_ds5.axf) file. Select Project | Build Project from the main menu. The results will appear in the Console tab (and the Problems tab if any errors occur). If all the changes have been entered as shown, there should be no errors.



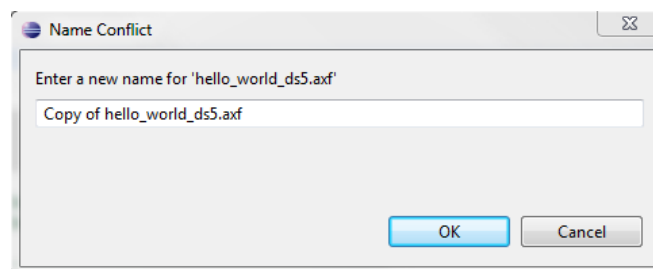
Boot the ZED Target using the ARM DS-5 Executable

The .axf file created in the previous section is an ARM eXecutable Format file, but it actually has the same layout as a standard .elf file created by the CodeSourcery compiler in the SDK. Since we retargeted the standard C library environment to utilize the BSP functions created for Zynq in the SDK, we can now simply substitute our DS-5 executable for the .elf file in our boot image.

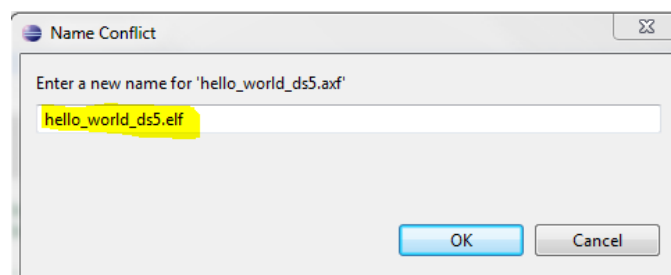
1. Make a copy of the .axf file and rename the extension to .elf, so that it will be recognized by the FSBL boot sequence.
 - a. In the Project Explorer tab, select the **hello_world_ds5** project, expand the **Debug** folder and select **hello_world_ds5.axf**.



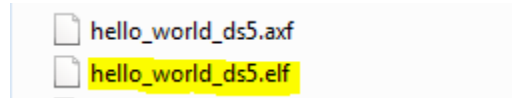
- b. Right-click on **hello_world_ds5.axf**, and select **Copy** from the drop-down menu. Select the Debug folder, right-click and select **Paste**. This will generate a name conflict with the original executable and a pop-up dialog will appear.



- c. Change the name in the text field to **hello_world_ds5.elf** and click the OK button.

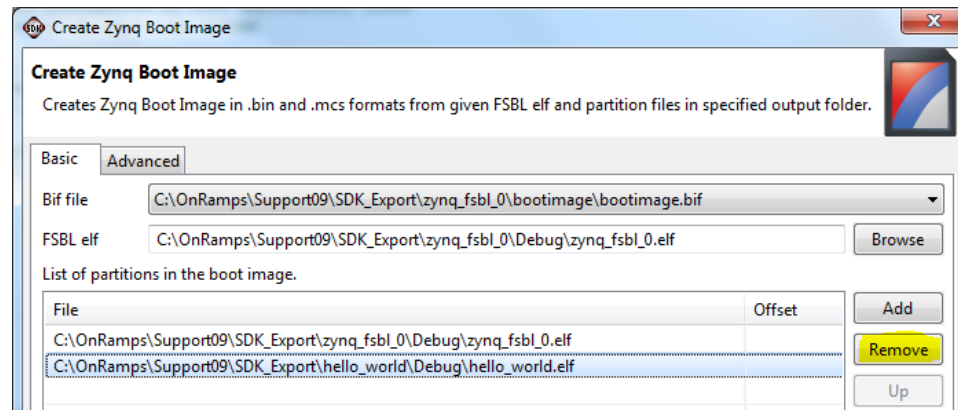


- d. You will now see both executable files in the Debug folder of the hello_world_ds5 project.



2. Return to the original SDK project (re-open SDK if you closed it earlier). Use bootgen to recreate the boot image, but replace the original hello_world.elf file with our DS-5 executable **hello_world_ds5.elf**.

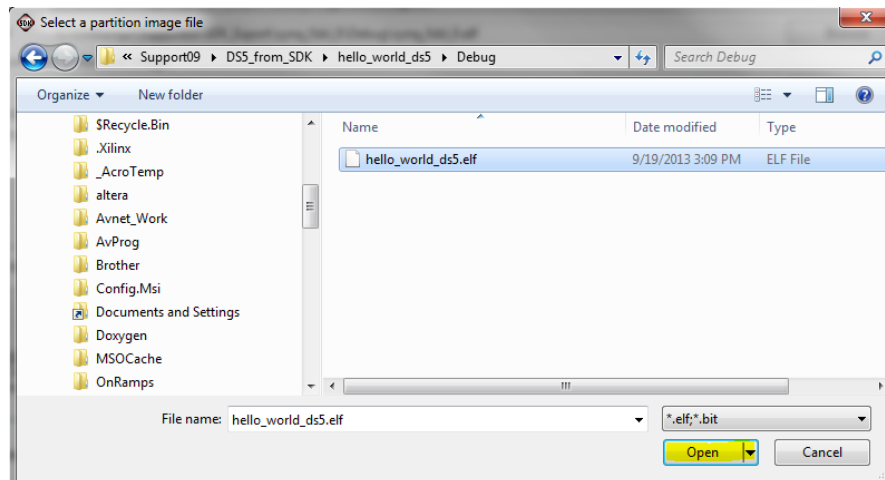
- a. In the SDK main menu, select **Xilinx Tools | Create Zynq Boot Image**. This will bring up the same populated dialog that was used to create the original boot.bin file. Select the **hello_world.elf** file and click the **Remove** button.



- b. Click the **Add** button. Browse to:

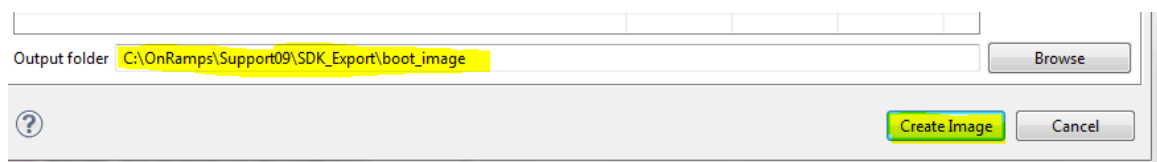
C:\OnRamps\Support09\DS5_from_SDK\hello_world_ds5\Debug

Select the **hello_world_ds5.elf** file and click the **Open** button.

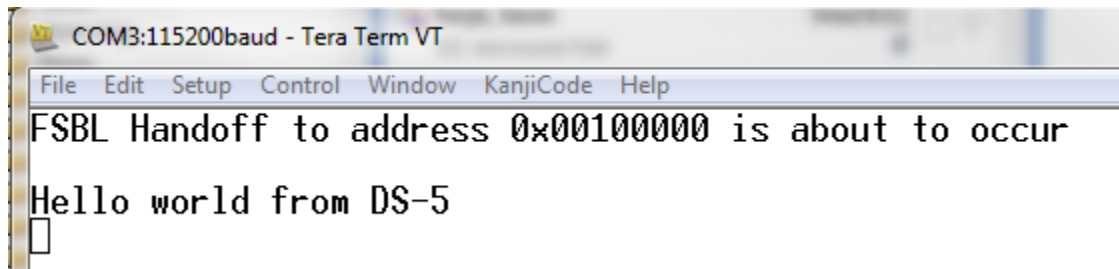


- c. For the Output folder, browse to the original location where you created the boot image and click the **Create Image** button.

C:\OnRamps\Support09\SDK_Export\boot_image



3. In the boot_image file, delete the **boot.bin** file we used to boot the ZED target originally. Rename the new image file **hello_world_ds5.bin** to **boot.bin** and use this to replace the existing boot.bin file on the SD card. Boot the ZED target and monitor the output in Tera Term Pro.



Now you have a DS-5 application project that can be expanded and built within the DS-5 environment, using the ARM compiler. When you wish to boot the ZED target, you need only follow the steps in this last section.

Although this is a very simple example that retargets only the UART, the same technique can be followed to retarget other required peripherals in the SDK environment that are needed by your application.

Revision History

Date	Version	Revision
16 Sept 13	00	Initial Draft
20 Sept 13	01	Release
10 Oct 13	02	Adjusted instructions to support MicroZed target

Resources

<http://www.zedboard.org>

<http://www.xilinx.com/zyng>

<http://www.arm.com/products/tools/software-tools/ds-5/index.php>