

# **ARM DS-5 Tools and Avnet ZED Series**

**#4**

## **Debugging Two Processors using ARM DSTREAM and Avnet ZedBoard or MicroZed**



Sept 2013  
Version 01

## Table of Contents

ARM DS-5 Tools and Avnet ZED Series .....	3
Required Installations .....	4
Technical Support .....	5
Debugging Two Cores Simultaneously.....	6
Create C Projects from Existing Files in DS-5 .....	6
Debug Two Cores Simultaneously .....	14
Revision History .....	22
Resources .....	22

## ARM DS-5 Tools and Avnet ZED 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](http://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](http://www.arm.com/support) .

## Debugging Two Cores Simultaneously

In this tutorial we will import two standalone bare metal projects into the DS-5 IDE, and use the USB-JTAG connection via the DSTREAM to download both applications, one to each of the ARM Cortex-A9 cores on the ZedBoard or MicroZed.

During this session, we will be using two programs that write to a common memory area in DDR3. We will boot the target from an SD card to initialize the Processor System (PS), and enter the DS-5 environment to import and debug our applications on the dual ARM CPUs.

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:

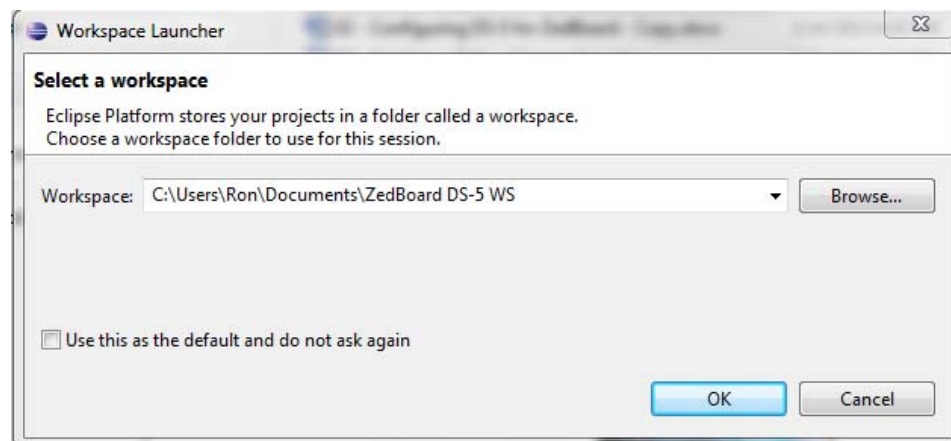
**<Download package folder>\Support04**

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

**C:\OnRamps\Support04**

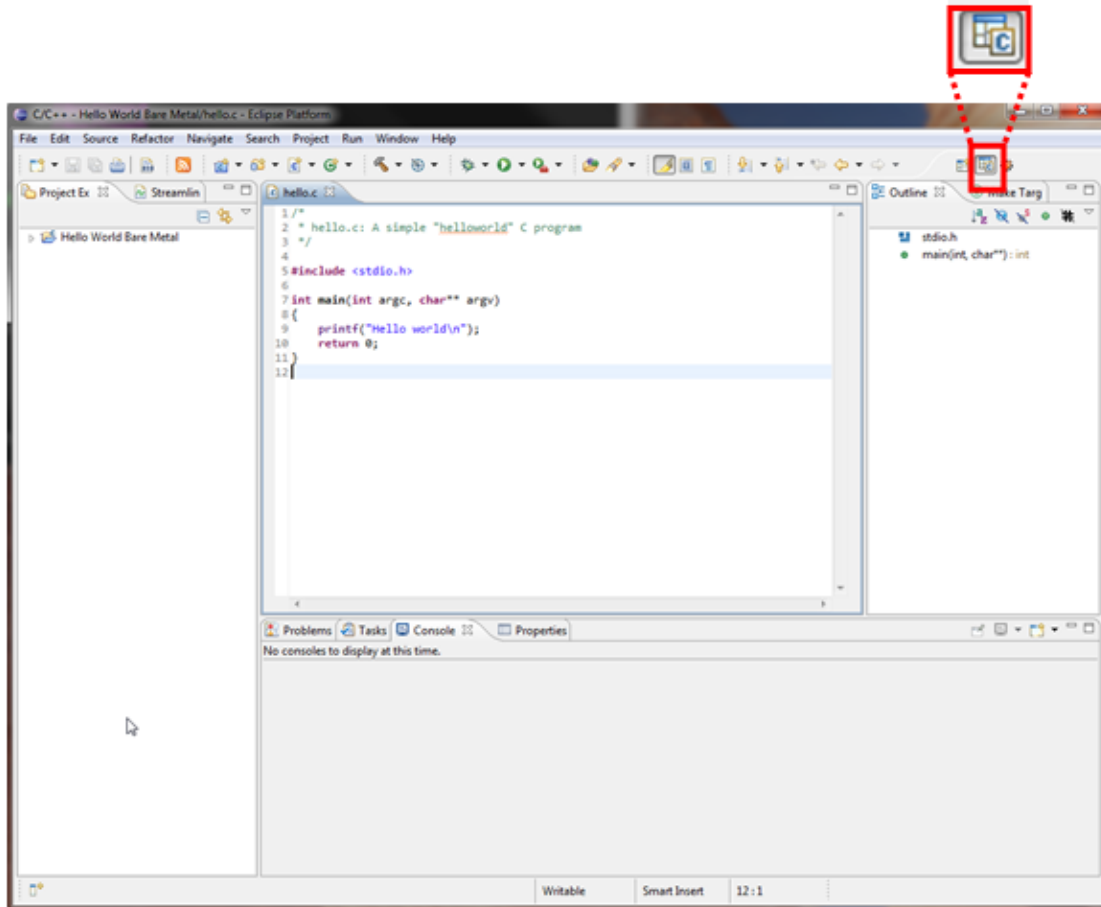
### Create C Projects from Existing Files in DS-5

1. Open the DS-5 IDE on your host PC. For the purposes of this tutorial series, we will continue to use the **ZedBoard DS-5 WS** workspace created in an earlier lesson.



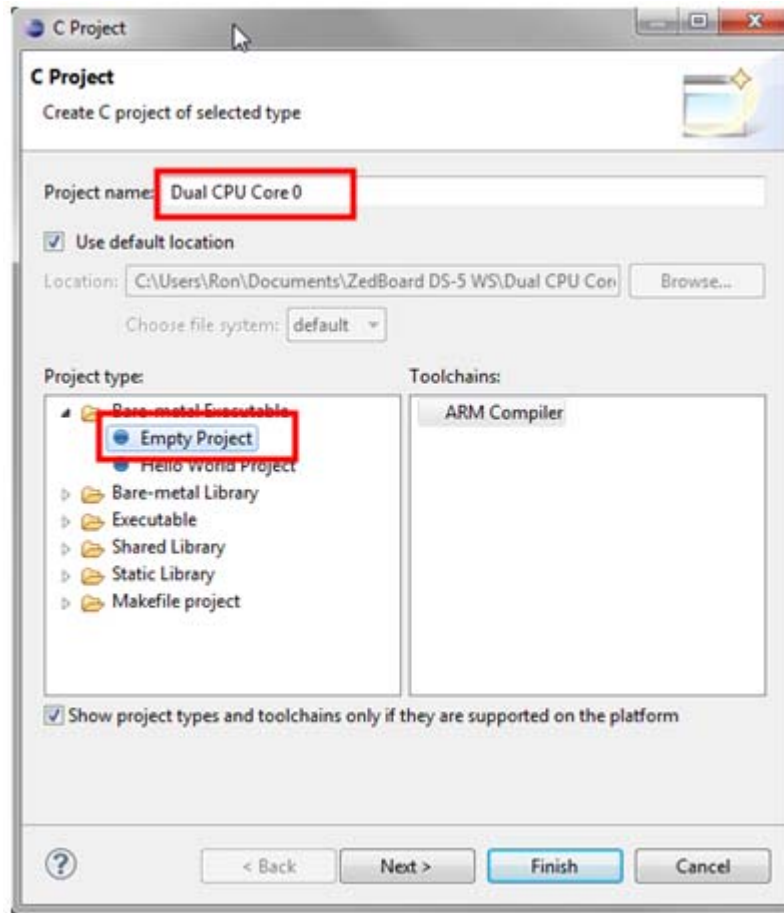
**Select a DS-5 Workspace**

2. If you have completed each earlier tutorial in the series, the C/C++ perspective of your workspace will appear as shown. If your workspace opened in another perspective, switch to the C/C++ perspective now by clicking on the icon indicated below.



**DS-5 C/C++ Perspective**

3. We will begin by creating a new software project in the C++ perspective. In the main menu, select **File | New | C Project**.

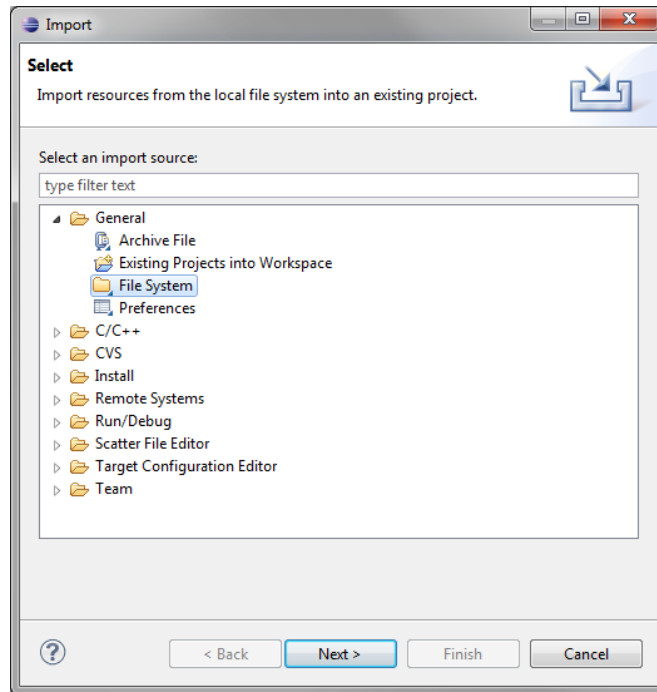


### New C Project Window

Name the project **Dual CPU Core 0** to indicate that we will be running this application on Core 0 in the Zynq chip, and select the **Empty Project** under Bare-metal Executable in the Project Type panel. By default, the ARM C Compiler will be used to generate the object files. Click the **Finish** button to generate both Debug and Release configurations.



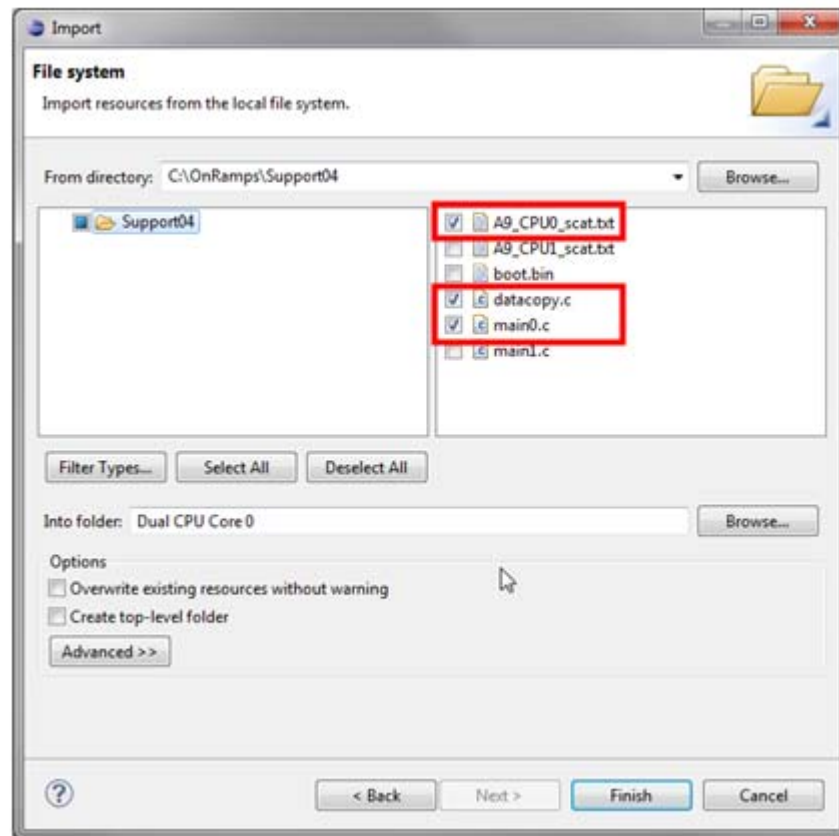
- Now we will import some source code to our empty project from the Support Files that were unzipped at the beginning of this tutorial. In the **Project Explorer** tab, right click on the new **Dual CPU Core 0** entry, and select **Import** from the drop-down menu.



### Select Files to Import

Expand the **General** folder, select **File System** and click the **Next** button.

5. Use the upper **Browse** button to locate the folder where you placed the support files. In the right-side panel, you will see a list of files to select. Click the check box next to the three files as shown (**A9\_CPU0\_scat.txt**, **datacopy.c**, **main0.c**):

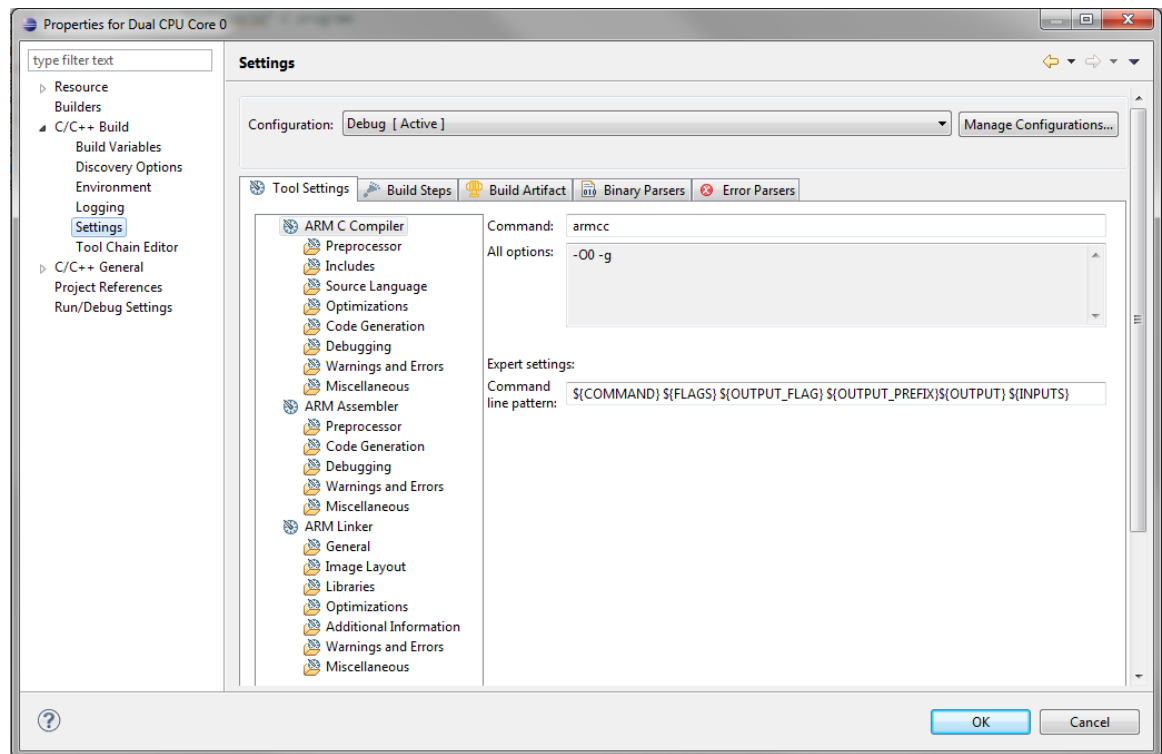


### Import Source Files to DS-5 Project

Click the **Finish** button to complete the import.

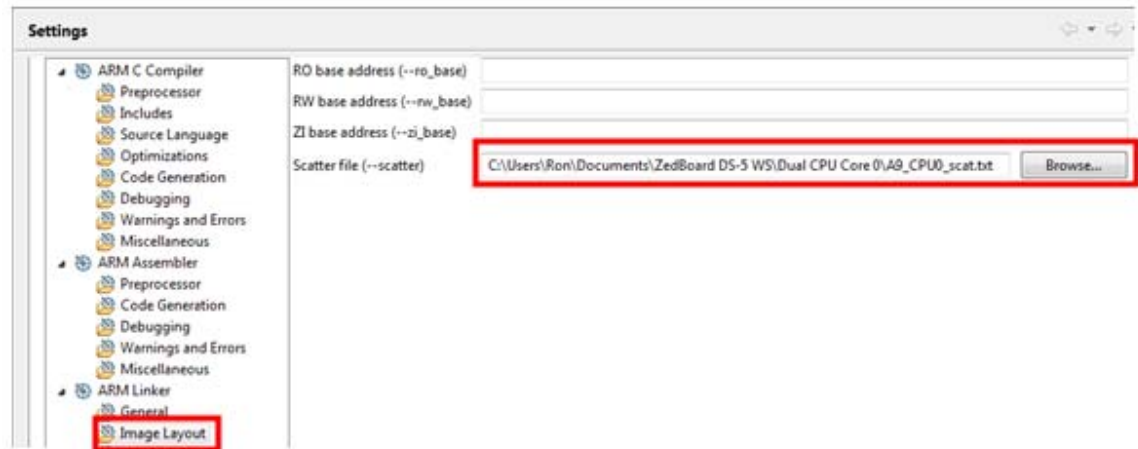
6. We are going to be running two applications, one on each core, that share a single memory space. We need to map the executables to specific locations in memory that will not cause the executables to overwrite each other at load time.

In the Project Explorer pane, right-click on **Dual CPU Core 0** and select **Properties** from the drop-down menu. Expand the C/C++ Build entry in the left panel and select **Settings**.



**Dual CPU Core 0 Build Properties**

7. Select **Image Layout** under the **ARM Linker**. Click the **Browse** button and locate the **A9\_CPU0\_scat.txt** file that you imported into your workspace.

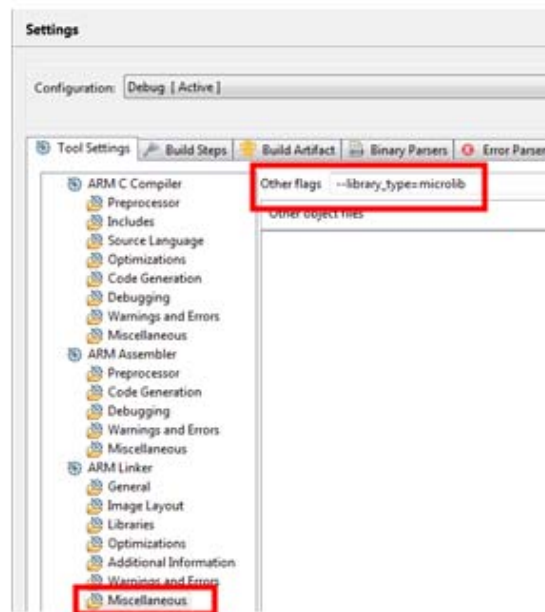


### Build Properties – Image Layout

8. The second change to the default Build Properties is to specify the library type. This is an optional library specification that causes the executable to be linked with a library that has been highly optimized for small code size.

Select **Miscellaneous** under the **ARM Linker**. In the **Other Flags** text box, enter:

**--library\_type=microlib**



### Build Properties - Miscellaneous

Click the **OK** button to save the Build Properties changes.

9. In the Project Explorer tab, click on the **Dual CPU Core 0** entry to ensure it is selected. From the main menu, select **Project | Build Project** to compile the source code. The code should build without errors and create an ELF file called **Dual CPU Core 0.axf**. If you expand the project, you will find this file in the Debug folder.
10. Now that we have our application set up for Core 0, we need to do exactly the same procedure for Core 1. Follow the same procedure from step 3 through step 9, with the following changes:

Project name: **Dual CPU Core 1**  
Import Files: **datacopy.c, main1.c, A9\_CPU1\_scat.txt**

When you have completed the second project, we are ready to begin a debug session on the board.

## Debug Two Cores Simultaneously

### For ZedBoard:

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

	<b>JP11</b>	<b>JP10</b>	<b>JP9</b>	<b>JP8</b>	<b>JP7</b>
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:

	<b>JP3</b>	<b>JP2</b>	<b>JP1</b>
Position	2-3	2-3	1-2

You should have your DSTREAM, ZED target, ZedBoard Adapter and host PC connected together as described in Tutorial #1.

To prepare to boot the target, copy<sup>1</sup>:

**C:\OnRamps\Support04\<ZED target>\boot.bin**

to the SD card, and insert the card into the SD card slot on the underside of the target. We will use a bare metal application built with the Xilinx SDK libraries to initialize the target in preparation for connecting the DS-5 Debugger<sup>2</sup>. In later tutorials, we will be working with Linux, so a logical application to use is the second stage boot loader U-boot. Since we do not have an OS to jump to at this point, we will want to interrupt our U-boot application before it tries to transfer control to the non-existent OS, which we

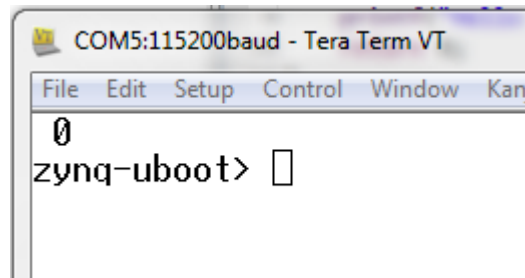
---

<sup>1</sup> You may have already copied this file if you followed Appendix I in Tutorial #2.

<sup>2</sup> The migration path from SDK to DS-5 is described in Tutorial #9 in this series.

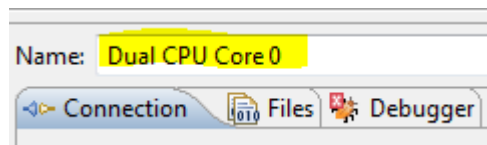
can do by monitoring the boot in Tera Term and hitting the space bar (or any key) on the host keyboard.

Power the target, start Tera Term<sup>3</sup> and interrupt the countdown process by hitting the space bar before it attempts to load the Linux kernel. If you aren't quick enough, you will get an error message indicating the ulmage was not found, but you will still be at the U-boot prompt as shown below. If you must power cycle the target, for the ZedBoard, the Cypress USB-UART driver will remain connected. But for MicroZed the SI Labs drivers will disconnect. You will need to disconnect Tera Term, then follow the same connection procedure by selecting the COM port chosen by the USB driver after every power cycle.



#### Interrupt the U-boot Countdown in Tera Term

1. In the Project Explorer tab, right-click on your **Dual CPU Core 0** project and select **Debug As | Debug Configurations** from the drop-down menu.
2. Right-click on the **DS-5 Debugger** entry and select **New** from the drop-down menu.
3. In the **Connection** tab:
  - a. Enter **Dual CPU Core 0** for the configuration **Name**.

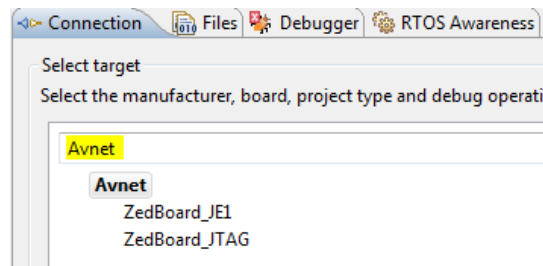


#### Configuration Name

---

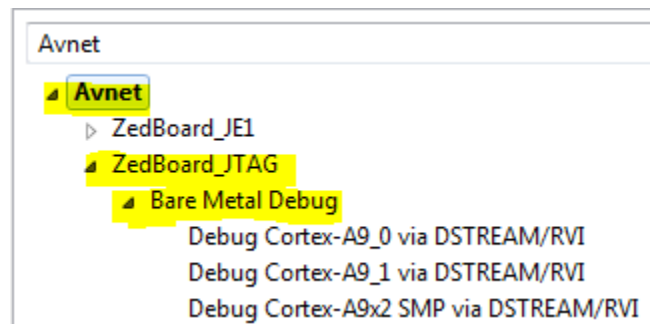
<sup>3</sup> We don't start Tera Term first because the Uart bridge must be active before Tera Term can detect the COM port.

- b. Type **Avnet** into the filter box in the **Select target** panel.



### Avnet ZED Target Configurations

- c. Expand the **Avnet**, **ZedBoard\_JTAG** and **Bare Metal Debug** entries as shown by clicking on the triangle to the right of the entries.

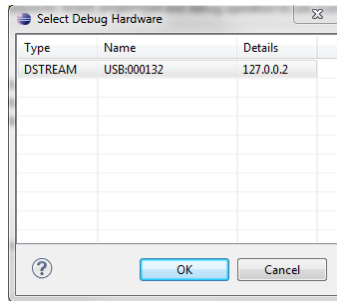


### Cascaded JTAG Configurations

- d. Select **Debug Cortex-A9\_0 via DSTREAM/RVI**. This selects processor 0 to run our application, and indicates we will be using the DSTREAM unit for debugging.



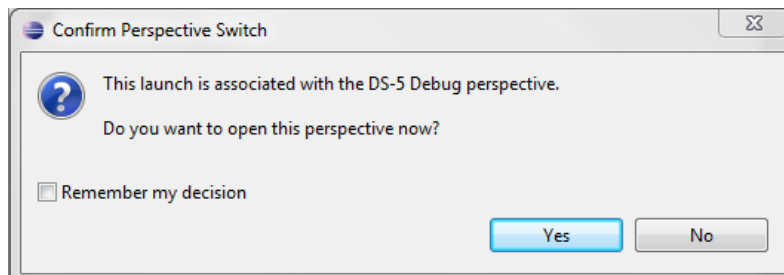
4. Still in the **Connection** tab, click on the **Browse** button in the **Connections** panel. In the pop-up window that appears, after a few moments you should see a USB connection to the DSTREAM unit. Select the DSTREAM entry and click **OK** to populate the Connection box.



### DSTREAM Connection

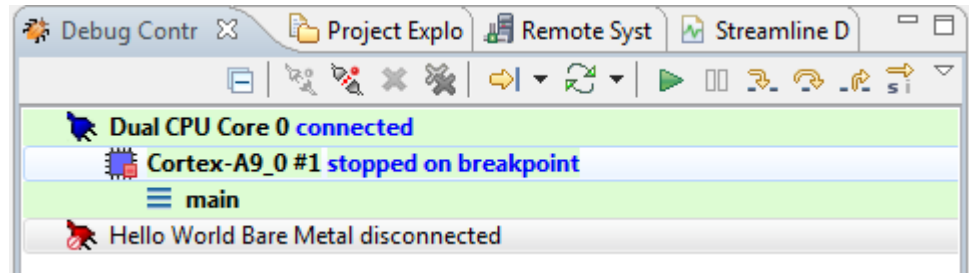
If you do not see an entry in the Select Debug Hardware window, check that your DSTREAM unit is powered on and is plugged into an active USB port on your host computer. Make sure all connections are made as specified in Tutorial #1.

5. Click on the **Files** tab. Under Target Configuration, click the **Workspace** button. Expand the **Dual CPU Core 0** and **Debug** entries by clicking on the triangle to the right of each entry. Select **Dual CPU Core 0.axf** and click the **OK** button to populate the **Application on host to download** text box.
6. Still in the Files tab, in the Files panel click the **Workspace** button. Use the same procedure as in the previous step to select **Dual CPU Core 0.axf** as the file containing the symbols for debugging.
7. Select the Debugger tab and click the **Apply** button. We want our application to stop as soon as the main entry point is reached, so we leave the **Debug from symbol main** selected.
8. Click the **Debug** button. We are still in the C/C++ perspective, so Eclipse asks if you want to switch to the Debug perspective. This will present a new set of windows specific to the debugging procedures, and since this is what we want click the **Yes** button.



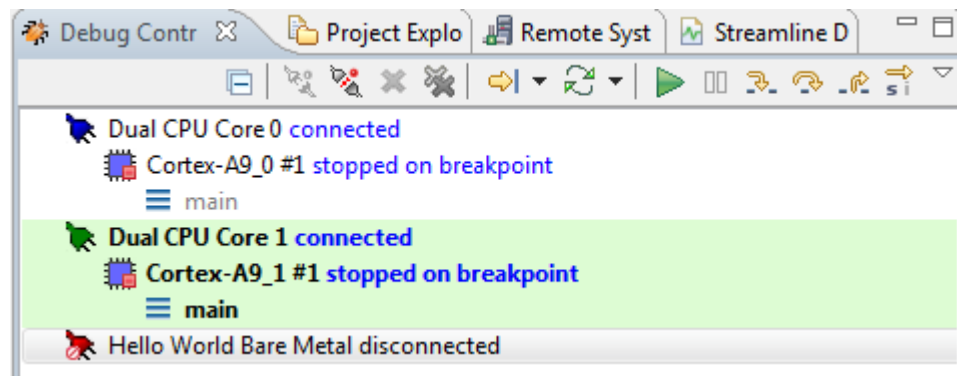
### Confirm Perspective Window

9. After a few seconds to connect and download the application, the Debug perspective will populate. You should note in the **Debug Control** tab at the upper left that our program has stopped on the entry breakpoint on processor 0.



### DS-5 Control Tab

10. Right-click in the **Debug Control** tab and select **Debug Configurations** from the drop-down menu. Create a second new configuration for the **Dual CPU Core 1** project, repeating the sequence you followed in steps 3 through 9. For the target, select **Debug Cortex-A9\_1 via DSTREAM/RVI**. When you are done, the Debug Control tab should appear as shown below.




### DS-5 Control Tab with Two Applications Stopped

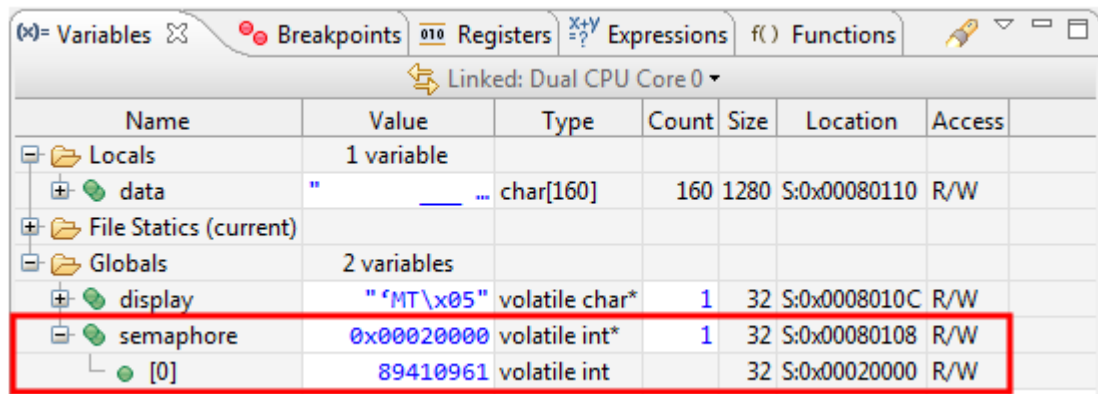
Select each stopped application in turn in the Debug Control tab, and notice that the Debug Perspective updates to reflect the active configuration. You will see in the source window that each application is stopped on a **for** loop, preceding a semaphore test. In **main0.c**, the semaphore tests for a 0 value, while in **main1.c**, the semaphore tests for a 1 value.

Although the applications are running on separate cores, they share various resources, such as the DDR3 memory. The semaphore is used to control access

to the memory, so that each application must hold the semaphore before the other may proceed.

11. Select the **Dual CPU Core 0** configuration to orient the Debug Perspective, and select the **Variables** tab in the upper right panel. You may resize the panel by moving the cursor to the double edges until the cursor changes to a double-headed arrow , then drag the edge as you like.

Expand the **Globals** entry and expand the semaphore variable. We can see in the debug display that semaphore is actually a pointer to a memory location at 0x20000, and that the value at that location is a random value, as DDR3 has not been initialized.

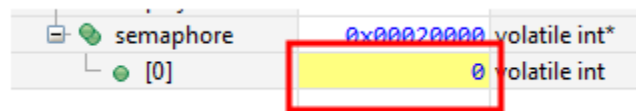


Name	Value	Type	Count	Size	Location	Access
Locals	1 variable					
data	"	char[160]	160	1280	S:0x00080110	R/W
File Statics (current)						
Globals	2 variables					
display	"MT\x05"	volatile char*	1	32	S:0x0008010C	R/W
semaphore	0x00020000	volatile int*	1	32	S:0x00080108	R/W
[0]	89410961	volatile int		32	S:0x00020000	R/W

**Dual CPU Core 0 Variables Tab**



If we were to start both applications, they would each simply spin on the for loops, as neither would be able to grab the semaphore for a memory write because the value is not 0 or 1. We can use the debugger to alter the program behavior on the fly.

12. With both applications stopped, select the **Value** column of the semaphore memory address [0] and replace the random contents with 0.



semaphore	0x00020000	volatile int*
[0]	0	volatile int

**Initialize Semaphore Value**

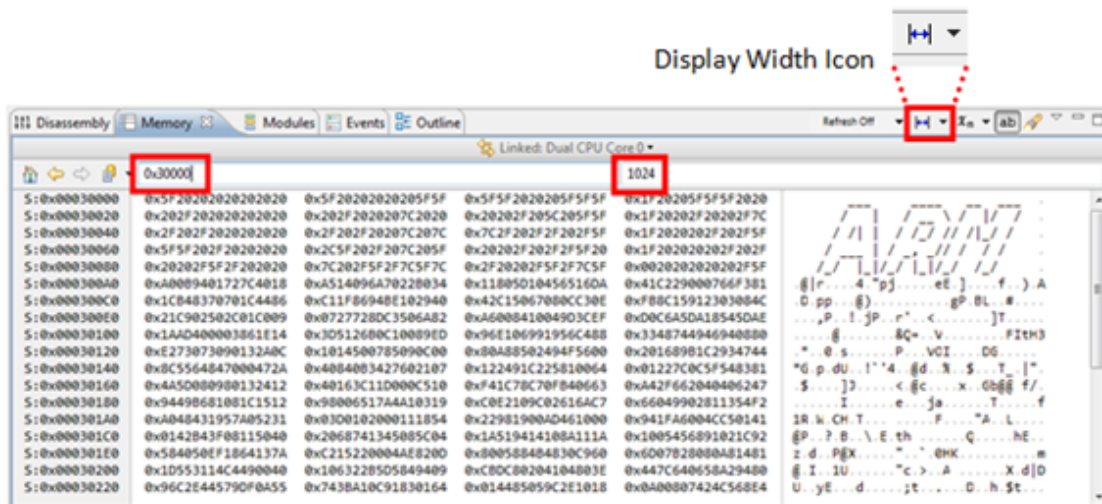
13. Using the debug controls, start the Core 0 application running . After a few seconds, pause  the application. Go back to the **Variables** tab, and you will see the semaphore value has changed from 0 to 1, indicating that the application on Core 0 has completed its memory write and has relinquished the semaphore.

We can see in the main0.c tab in the Source panel that there is display area assigned a memory address of 0x30000. Let's examine the memory at this location to see what our application has done.

```
3 volatile int *semaphore = (int *)0x20000;  
4 volatile char *display = (char *)0x30000;  
5
```

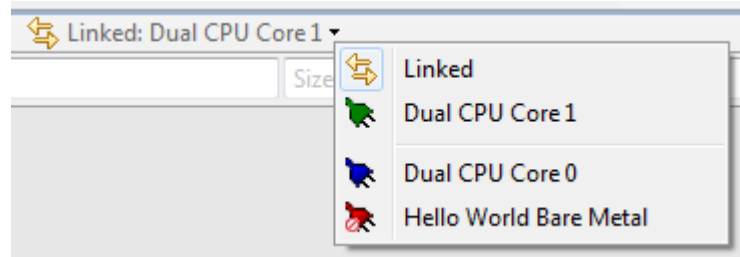
### Display Buffer

14. Select the Memory tab in the low level panel (lower right) and enter a memory address of **0x30000** and the number of bytes to **1024**. Click the down arrow next to the **Display Width Icon** and select 8-bytes. Finally, drag the left edge of the panel to the left to expand the memory display until you see the view below.



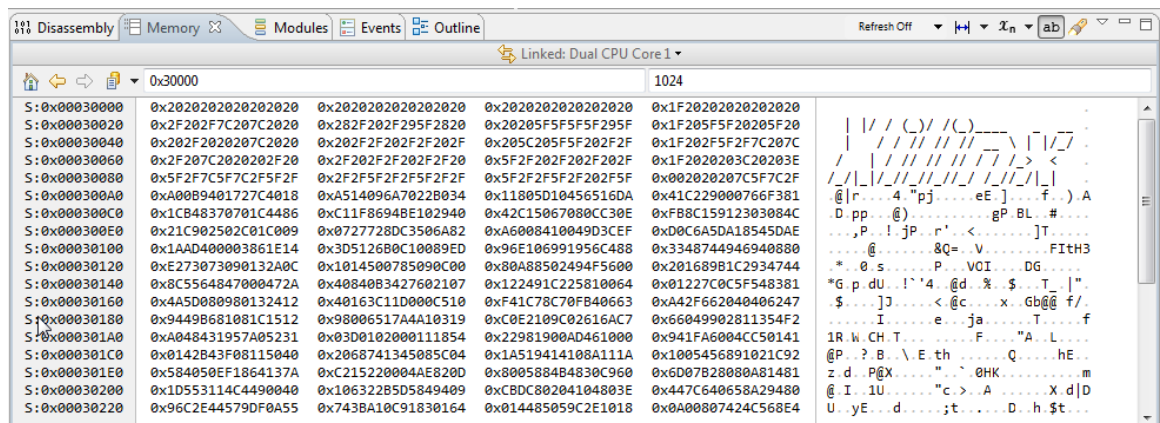
### Memory Tab After Core 0 App Write

15. Start the Dual CPU Core 1 application running. The Memory panel blanks as the view is automatically switched to the Core 1 configuration, and we haven't set any view parameters here yet. If you wished to lock a particular view to a configuration, you may do so by selecting the drop-down list in the Linked field at the tops of the panels. For our purposes, we will leave the panel linked, to follow the active application.



### Lock View to a Debug Configuration


Pause the Dual CPU Core 1 application, and configure the Memory window in exactly the same way as we did for the Core 0 application. You will see the view shown below.



### Memory Tab After Core 1 App Write

16. You may stop and start both Core configurations independently, and run both at the same time if you wish. Use the single-step debug controls to explore the applications and determine how the memory writes and semaphore interaction works.

This has been a brief introduction to the DS-5 Debug perspective and how it can be used to independently debug applications running simultaneously on each ARM core. More advanced topics will be covered in subsequent tutorials in this series.

You can disconnect the current debug sessions , close DS-5 and power down the hardware.

## Revision History

Date	Version	Revision
23 May 13	00	Initial Draft
20 Sept 13	01	Release

## Resources

<http://www.zedboard.org>

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

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