

ARM DS-5 Tools and Avnet ZED Series

#8

Advanced Linux Application Debug using DS-5 and Avnet ZedBoard or MicroZed



September 2013
Version 01

Table of Contents

ARM DS-5 Tools and ZedBoard Series	3
Required Installations	4
Technical Support	5
Debugging a Linux Application.....	6
Boot the Target	7
Configure a Remote System Explorer (RSE) Connection	8
Start the Xming X-Server.....	11
Create Gnometriz Debug Configuration	12
Gnometriz Debugging in Detail with DS-5	16
Solution	23
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 .

Debugging a Linux Application

In this tutorial we will boot the target board with the same Linux kernel and root file system used in tutorial #7, and then download and debug an application.

If you have completed tutorial #7 (Streamline), the software setup here is exactly the same. Detailed instructions for creating the required software environment on both the host platform and ZED target are contained in the Appendices of tutorial #7, so it is highly recommended that tutorial #7 be completed prior to this tutorial.

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

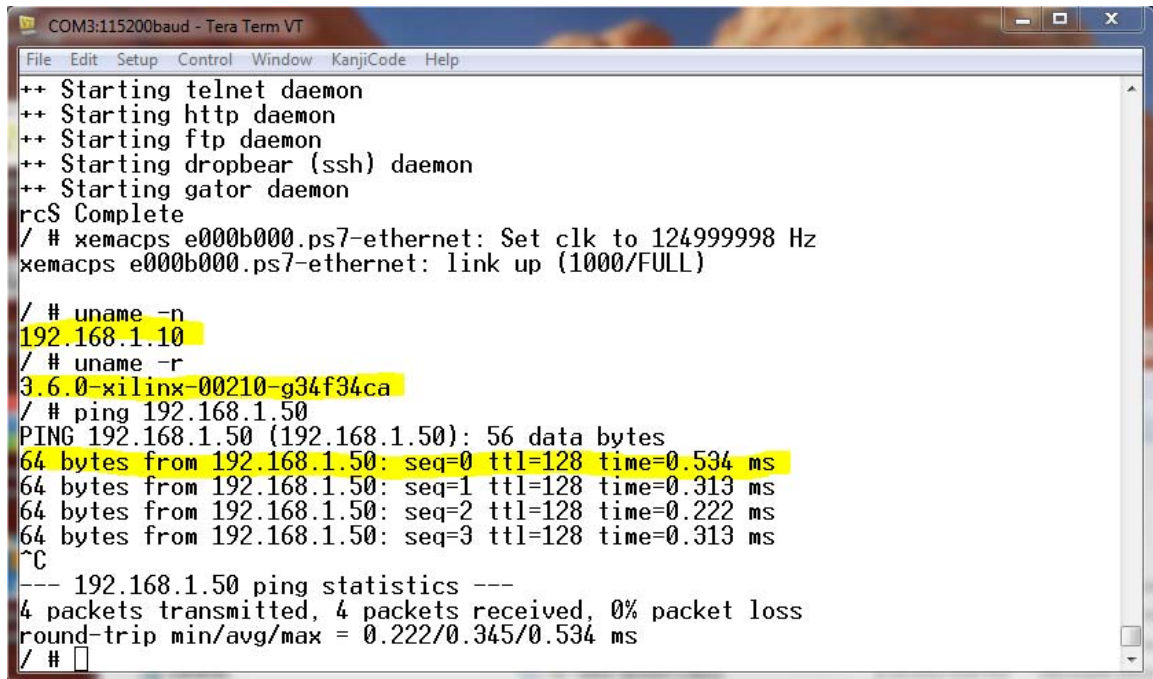
Connect an Ethernet cable and micro-USB cable between the ZED target and your host PC, as per previous tutorials. The host address should be on the same subnet as the ZED target, which has an address of **192.168.1.10**. For this tutorial, the host address is assumed to be **192.168.1.50**.

This tutorial assumes you already have your SD Card loaded with files from tutorial #7. If you have not already done so, insert the card into the SD card slot on the underside of the target.

Boot the Target

On power up, the target will initialize the PS subsystem, configure the PL, load and run the FSBL and load and run U-boot. U-boot will then load the Linux components and boot the target to a Linux command prompt. Use the following procedure:

1. Power the target.
2. Wait for the blue configuration LED to illuminate. Start Tera Term Pro on the host.
3. Wait for the boot to complete. You can verify the target is fully operational by entering the following commands at the prompt in Tera Term Pro:
 - a. **uname -n**
 - b. **uname -r**
 - c. **ping 192.168.1.50** (substitute the IP address of your host computer)
 - d. **Ctrl-C** to terminate the ping operation



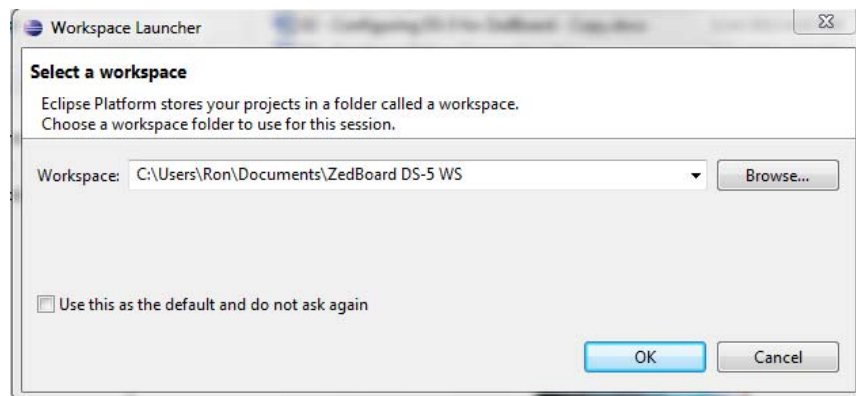
```
COM3:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
++ Starting gator daemon
rcS Complete
/ # xemacps e000b000.ps7-ethernet: Set clk to 124999998 Hz
xemacps e000b000.ps7-ethernet: link up (1000/FULL)
/ # uname -n
192.168.1.10
/ # uname -r
3.6.0-xilinx-00210-g34f34ca
/ # ping 192.168.1.50
PING 192.168.1.50 (192.168.1.50): 56 data bytes
64 bytes from 192.168.1.50: seq=0 ttl=128 time=0.534 ms
64 bytes from 192.168.1.50: seq=1 ttl=128 time=0.313 ms
64 bytes from 192.168.1.50: seq=2 ttl=128 time=0.222 ms
64 bytes from 192.168.1.50: seq=3 ttl=128 time=0.313 ms
^C
--- 192.168.1.50 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.222/0.345/0.534 ms
/ #
```

Linux Command Line Results on Target

Configure a Remote System Explorer (RSE) Connection

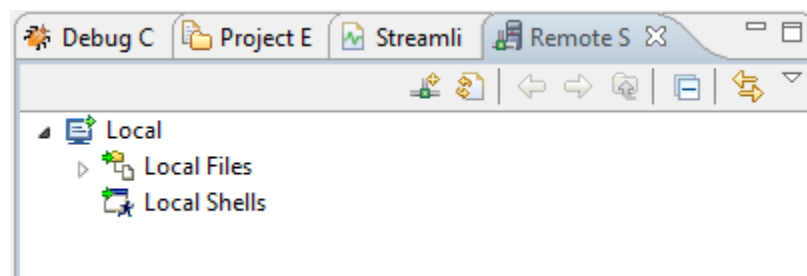
Debug communication with the ZED target will be done over an Ethernet cable between the target and our host. Before we can do this we must inform DS-5 how to configure the connection. If you have already completed tutorial #7, you may skip to the next major section.

1. Open the DS-5 IDE on your host PC. For the purposes of this tutorial, we will use the workspace name **ZedBoard DS-5 WS**, but any name/location you choose is fine (including the default workspace).



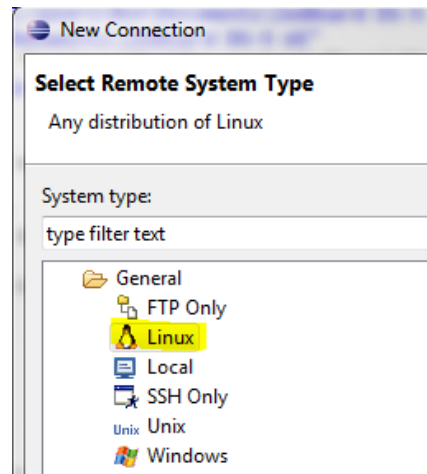
Select a DS-5 Workspace

2. If you do not have a Remote System tab in the panel with Project Explorer, from the main menu select **Window | Show View | Other | Remote Systems | Remote Systems**. Click the OK button and select the **Remote Systems** tab.



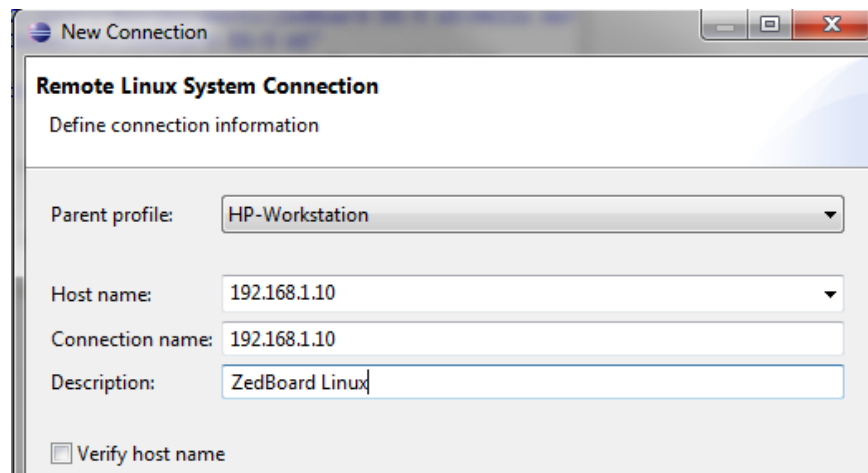
DS-5 Remote Systems Tab

3. Anywhere inside the panel, right click and select **New Connection**. In the **New Connection** window, select **Linux** and click the **Next** button.



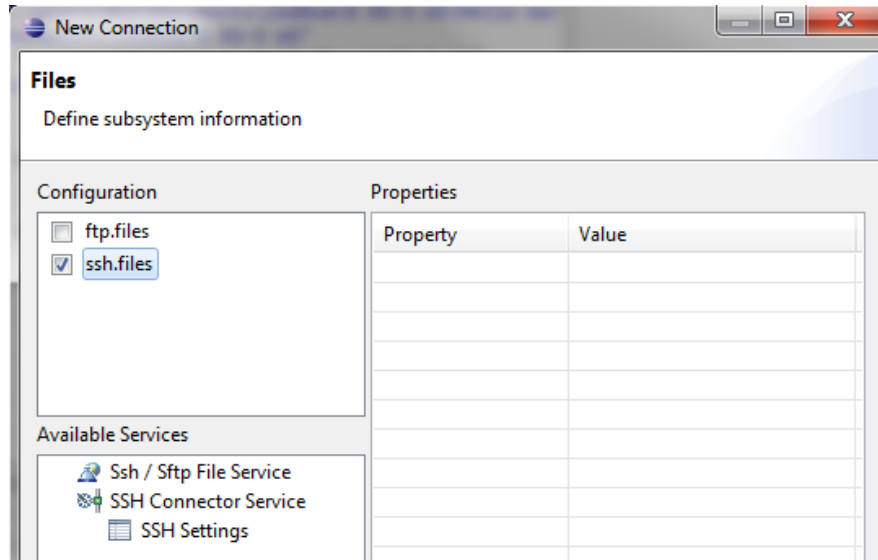
Create a Remote System Connection for Linux

4. In the Host name field, enter the target IP address of **192.168.1.10**. This will be copied into the Connection name field. You can optionally enter a description if you wish. The **Parent Profile** name will be dependent on your host platform, but leave it unchanged. Click the **Next** button.



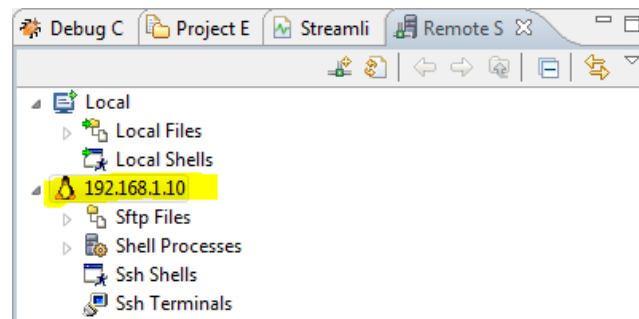
RSE Connection Parameters for the ZED Target

5. Select the **ssh.files** configuration (Secure Shell), as we will use this protocol to communication with the target instead of FTP. Click the **Finish** button.



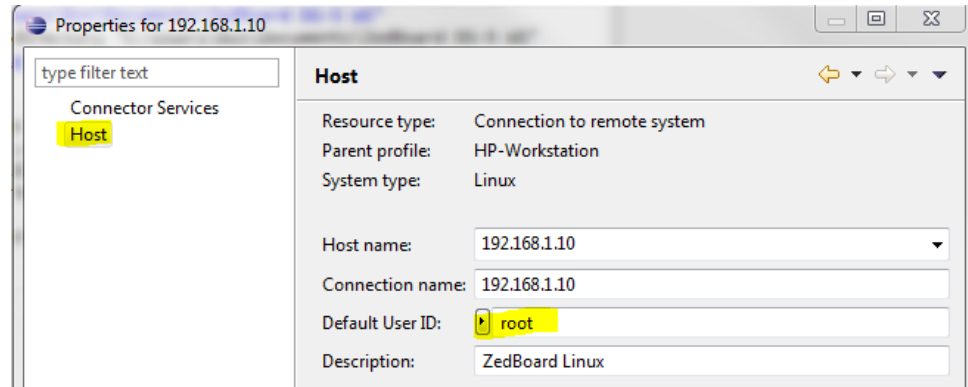
Secure Shell Configuration

6. In the Remote Systems tab, right-click on the new entry for the target IP address and select Properties.



New Remote System Configuration for Target

7. Select the **Host** entry and click the arrow next to the **Default User ID**. Enter **root** for the ID and click the **OK** button.




Create root Login for ZED Target

Now the Remote Systems configuration for the ZED target is complete, we can use this in the future to establish a debug connection for any Linux application project in DS-5.


Start the Xming X-Server

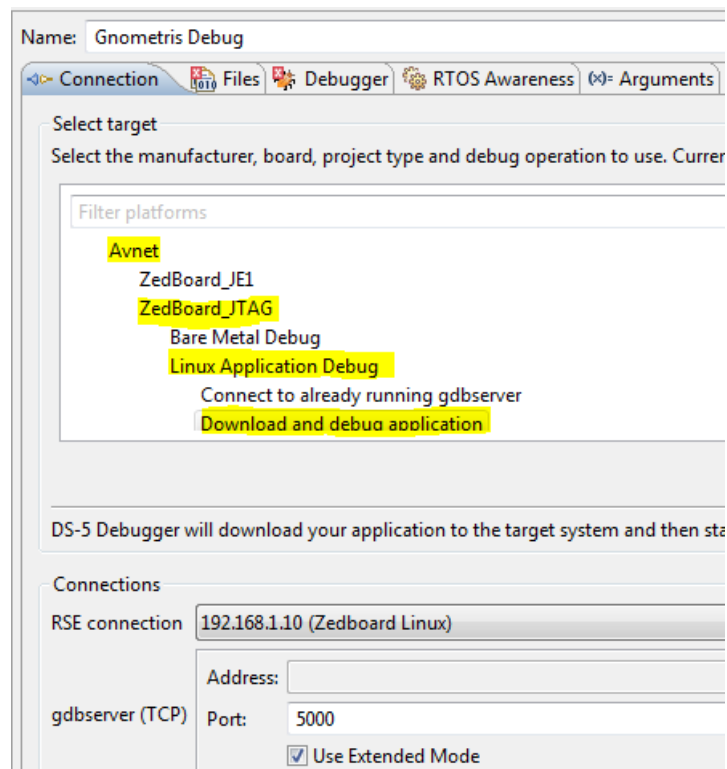
You should already have created a configuration file called **config.xlaunch** in tutorial #7. If not, you can follow the detailed instructions in Appendix III of Tutorial #7 to install and configure the X-Server.

1. Double-click on the **config.xlaunch** file on your host computer. You will see a small icon appear in the system tray to indicate that the server is running and ready to accept connections. 

Create Gnometris Debug Configuration

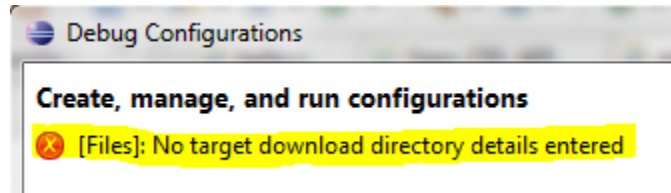
Gnometris is an open-source Linux implementation of the well-known Tetris game. If you have completed the set-up requirements, you will already have a **gnometris** project in the C/C++ perspective of DS-5. In order to load, run and debug the application we must inform DS-5 of the connection parameters between your host computer and the target Linux platform.

1. In the **Project Explorer** tab, right click on the **gnometris** project and select **Debug As | Debug Configurations** from the drop-down menu.
2. In the left Debug Configurations panel, select **DS-5 Debugger** and click on the New Configuration  button.
3. Enter **Gnometris Debug** for the configuration **Name**.
4. In the filter text box, enter Avnet to show only the configurations relevant to our target. Move your mouse cursor inside the panel and click on the arrow that appears to the left of **Avnet** to expand it. Also expand **ZedBoard_JTAG** and **Linux Application Debug**. Select the **Download and debug application** operation.



Gnometris Connection Configuration

5. Notice that DS-5 is aware that the RSE connection established earlier is the correct connection to use for this debug session. It automatically selected the correct IP address and port (5000) to use. Also note that the Debug button is still grayed out. DS-5 does not activate the button until all required configuration items are filled in, and displays the current known issue near the upper left of the window.

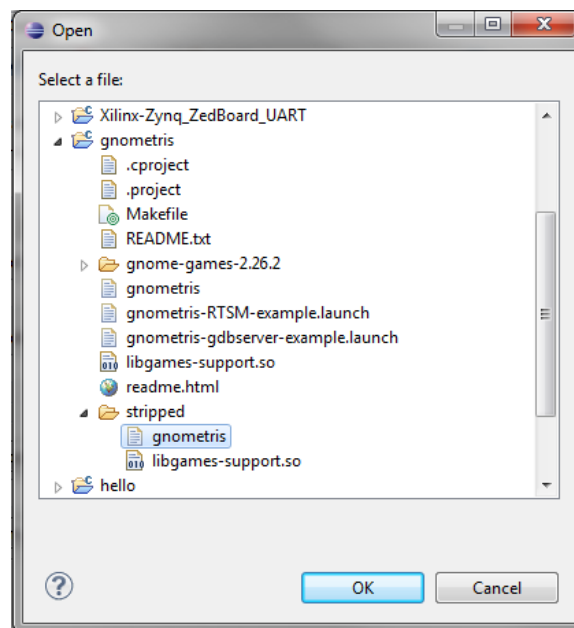


DS-5 Issue Report Line

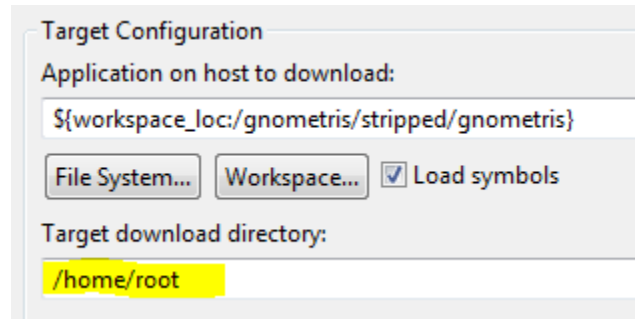
In this case the issue indicates that the Files tab does not have a target download directory specified, so select the **Files** tab to complete the information.

6. In the **Target Configuration** section, we need to tell DS-5 what application executable file to download. We do not need debug symbols on the target, as this association is made on the host computer (within DS-5). We can therefore use the stripped version of our gnometris application, which we will find in the DS-5 workspace.

Click the topmost **Workspace** button and select **gnometris | stripped | gnometris** for the application to download. Click the **OK** button to populate the text box.



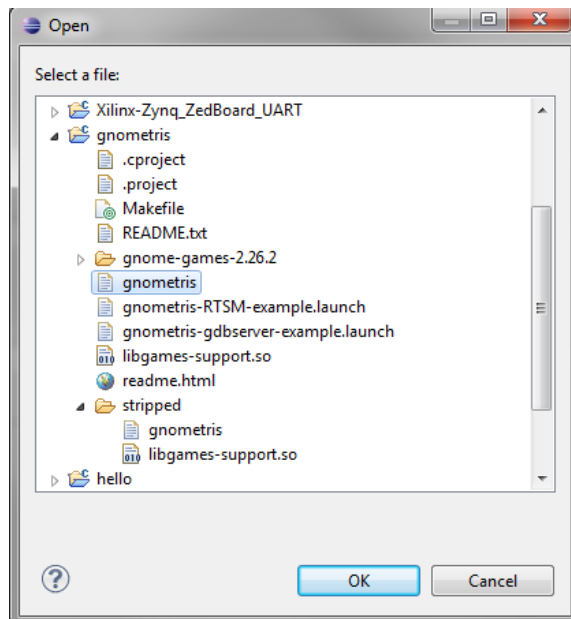
- Next, we need to specify the **Target download directory** as reported in the issues line. We must use a writable location on the target, which in this case is the directory of the root user we will be using to log in. Enter **/home/root** in the text box.



The **Debug** button is now available, indicating that the minimum requirements have been met, but there is still more to do for an optimal debug session.

- In order to associate the source lines with the executable code running on the target, DS-5 needs to be told where to find a file with symbol information. This is contained in the Debug version (unstripped) of the gnometris application. In addition, and this would be known to the programmer even if it is not apparent in our brief tutorial, the gnometris program requires an additional library that is not part of the standard linux system. The library will be loaded at runtime and we must tell DS-5 where to obtain it.

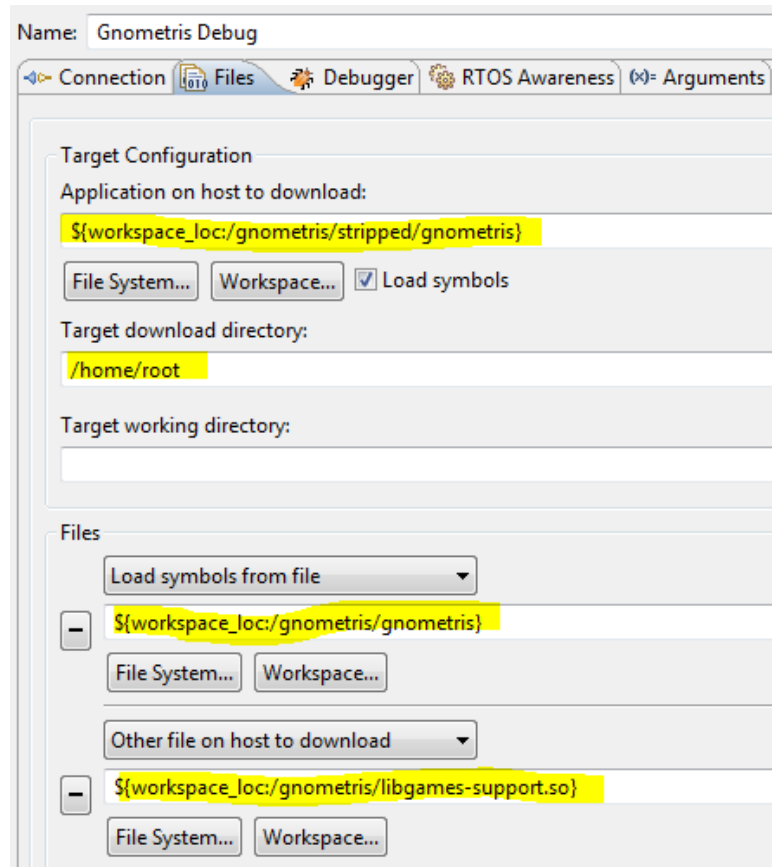
Click on the lower **Workspace** button. Select **gnometris | gnometris** and click the OK button to populate the text box.



- Next, click on the + button and click the new **Workspace** button below the **Other files on host to download** button. This is where we will add the required library.

Select **gnometris | libgames-support.so** and click the **OK** button to populate the text box.

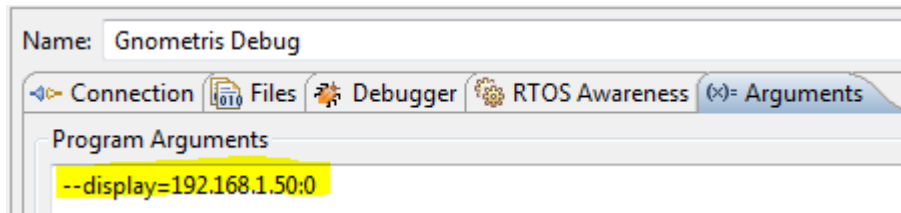
Your Files tab should now appear as shown below:



DS-5 Files Tab

10. There is one more application specific detail to take care of, and that is to tell gnometris where it should send its display output. We can have DS-5 communicate this information to the application via the Arguments tab.

Click on the **Arguments** tab. Enter the display parameter as shown in the diagram below (substitute your host IP address if it is different). Don't forget the **:0** at the end! (It specifies the number of the X-Server to use, in the event we have multiple X-Servers running on the host).



DS-5 Arguments Tab

Click the **Apply** button to save the configuration, and then click the **Debug** button. If you are asked to switch to the DS-5 Debug perspective, click the **Yes** button. DS-5 will initiate the debug connection to the target, download the application, execute it with the arguments we specified, and start the debug session using the parameters in the Files tab. If you are prompted for a password, enter "root" and click **OK**.


The application will stop on the main entry point. In the **main.cpp** source window, you will see the application halted on **line 43**.



Gnometris Debugging in Detail with DS-5


If you have used the Xilinx SDK in the past, much of what you see in the DS-5 debug perspective should look quite familiar. This is because both implementations are based on Eclipse, an open-source framework for the creation of development tools. Rather than simply concentrating on the basic aspects of debugging (single-stepping, setting breakpoints, looking at variable values), all of which is available in every source level debugger, we will try to focus on the features that differentiate the DS-5 debugger interface from the one you are used to in the SDK.


As stated earlier, Gnometris is an open source implementation of the old Tetris game. There is no graphical output as yet, because the application was halted by the debugger before the main window has been activated. You will get a chance later in this session to see the game operate via the X server, and play the game running on the target.

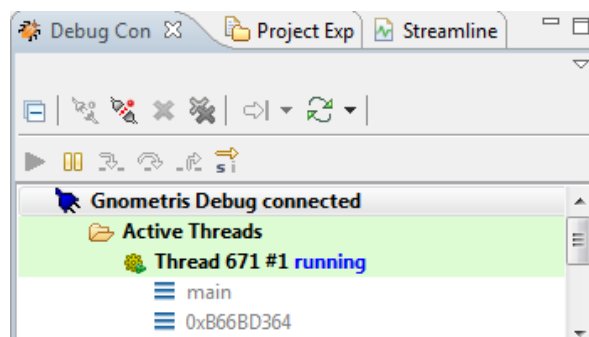
1. We will begin by setting a breakpoint in the function that rotates shapes as they "fall" down the display, called `BlockOps::rotateBlock`. Click on the **Functions** tab

and to quickly locate the function click on the Search button . Begin typing the name into the text field, until the function is located, and then click the **OK** button. **Double-click** on the function name to set a breakpoint at the first executable line of code. You will see the breakpoint indicator turn red, and you can check that the breakpoint is set in the **Breakpoints** tab.

While we are in the Breakpoints tab, notice that the buttons along the top change with each tab selection. You can discover the operations of active buttons in any tab by hovering the mouse cursor over the button. In the Breakpoints tab, the  button skips all active user breakpoints (not internal breakpoints set by the debugger, which cannot be skipped or disabled). Click on this button to see the breakpoint symbol next to our single breakpoint change, and notice in the Commands tab to the left that each action is recorded with the command line that drives the DS-5 function. The command history can be recorded and saved as a script to automate repeated operations. Click the  button again to re-enable the breakpoint.

Select our **rotateBlock** breakpoint, and click on the  button. This will open the file containing the breakpoint in the source window, and is a handy way to quickly view the code in question.

2. We want to set one more breakpoint in our code, this time in the **BlockOps::generateFallingBlock** function, at line 297. This time, click anywhere in the blockops-noclobber.cpp tab in the Source window and hit <CTRL>-L. Enter 297 in the pop-up window and click the **OK** button to go directly to the line. **Double-click** in the left margin of line **297** to set a breakpoint here in the conventional fashion.
3. In the Debug Control tab, click the run  button to start the application. An X window containing the game console will open. Notice in the Debug Control tab the call stack shows that the application is running.



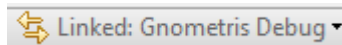
Gnometris Application Running in Debugger

4. Click in the console to set the focus to the X-Server and enter <CTRL>-N to start a new game. The Debug Console indicates that the program has hit a breakpoint, which you can confirm in the Source window.

Take a few minutes and peruse the contents of the following tabs, performing the suggested operations to get a better feel for the operation of the DS-5 debugger. In some cases it will be exactly the same as the SDK operation, while in other cases there are additional capabilities not found in the SDK debugger

Variables:

- a. **Highlight** the **posx** line (expand **this** to see it) and right-click to show the context menu. Change the representation to different formats.
- b. Select **Show in Memory**, and notice the contents of the Memory tab change.
- c. Select **Show in Disassembly** to go directly to the assembly line where the value is loaded.
- d. Take note of the **Location** column and select the value of the variable contained in a register.
- e. Select **Send to | Expressions** View to capture the variable for quick reference during execution.
- f. Note the **Linked** button contained in many of the tabs. When multiple debug sessions are active concurrently, this button can be used to keep the contents locked to a particular debug configuration, rather than toggling to the active session.




We have only a single active session here, so we won't be using the link feature, but this is very useful for multi-core debugging.

Registers/Memory/Disassembly:

- a. Open the **Registers** tab and locate the value contained in the register variable you viewed in the Variables tab. Confirm it is the same value.
- b. Scroll down to the program counter (PC). Single step the program a couple of times and note that the value changes, and all changed values are color highlighted. Toggle back to the **Variables** tab and note the highlighting also occurs for changed values there.
- c. Back in the **Registers** tab, highlight the PC line, right-click and select **Show Memory Pointed to by PC**. The **Memory** tab will display the code lines at that location (in Hexadecimal).
- d. Right-click again and select **Show Disassembly Pointed to by PC**. The Disassembly tab shows the code lines in ARM assembler.
- e. Click on the Memory tab and drag it into the lower panel. This move feature is available for many of the tabs and allows you to customize the display as suits your needs at the moment. With both the Disassembly

and Memory views open, you can monitor the operation of the program in several formats at once. Useful when debugging problems that may be related to hardware glitches in memory.


- f. Put **\$sp** in the **Address** field of the **Memory** tab and put **128** in the **Size** field. Hit the **Enter** key. This is the top of the stack (the stack grows downward into lower addresses. This is a very useful shortcut to see the stack contents when a program is misbehaving.

If at any point you inadvertently change something that affects program operation and causes a crash or exit, you can restart from the main entry point by clicking the restart  button in the Debug Control tab. Also, if you wish to restore all panels to their default positions and sizes, select **Window | Reset Perspective** from the main menu.

5. Use the skip breakpoints button (remember?) to temporarily disable all user breakpoints in the program. Now you can play the game for a little bit to give yourself a break and see how the game operates on the embedded platform.

Click the run button and make the X window your current view. You will see a new block created and start to “fall” down the game console. The arrow keys on your host keyboard control the operation:

- a. **Left arrow:** Move block one position left
- b. **Right arrow:** Move block one position right
- c. **Up arrow:** Rotate block 90 degrees
- d. **Down arrow:** Accelerate fall rate

When you have had enough play time, click the pause  button to stop the game at the current execution line. Re-enable all the breakpoints, but disable the breakpoint on line 297 by unclicking its select box. You should have only the internal breakpoint and the user breakpoint in the **rotateBlock** function active.

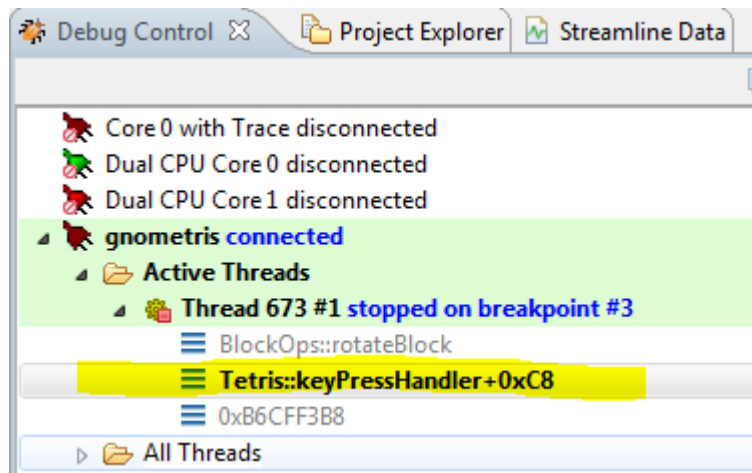
6. Click the start button, set focus to the X window and hit the **Up arrow** key to rotate the block. This will cause the program to hit the breakpoint we set earlier in the block rotate function.
7. If you want to quickly examine variable values in the Source window, without having to open the Variables tab, there is a temporary version of the Expressions tab called the Expression Inspector. Select a variable by double-clicking on it in the source, right-click and select **Inspect**. The Expression Inspector will pop up with the variable information. You can add other variables by typing in their names, or using drag and drop. Try this in the rotateBlock function by selecting

posx to open the Expression Inspector, then add posy and blocknr to the window. You can close the Expression Inspector when you are done.

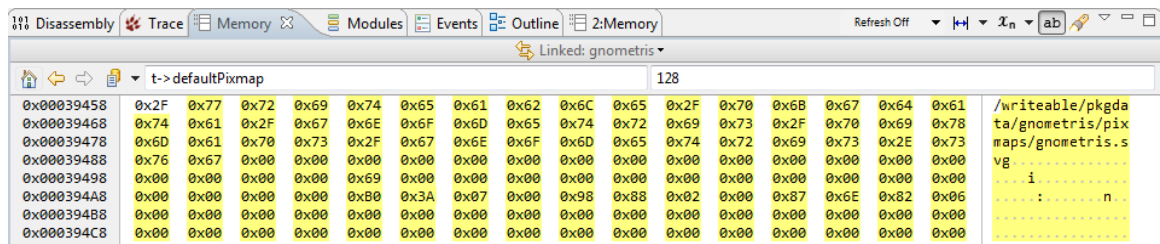
8. The Memory tab displays memory in various formats. It has an Address field which is an expression for the starting address and a Size field for the number of bytes to show. We tried this out earlier when we put the stack pointer (\$sp) into the address field.

You can also set the Address field by dragging selections from other views, like Variables or Expressions, and dropping it into the Memory view. We can try this to display the contents of the local variable t->defaultPixmap.

In the **Debug Control** tab, select the **Tetris::keyPressHandler** frame.



Now in the Variables tab, which has changed to represent the new frame selected, locate the member **t->defaultPixmap**. Select the row, right-click and select **Show Dereference in Memory**. Change the Memory size to **128**.



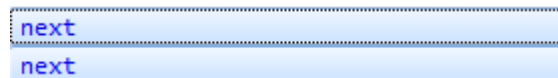
The Hex and character values in memory for the array contents are shown in the memory window.


9. You have likely noticed that debugging actions are captured in the **Commands** tab along with the responses. Commands are also recorded in the History tab. You can control the debugger from the command line through the **Command** text box in the Commands tab.

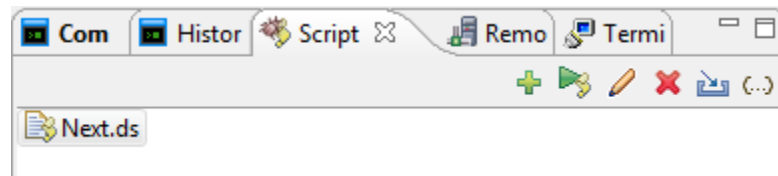
Type **set st** into the textbox in the Commands tab, then hold down the <CTRL> key and hit the space bar. This provides context-sensitive help, and can be used in many other places, such as in the Address field of the Memory tab. Hit the <ESC> key to remove the pop-up help window.

The next command is used to single-step the program. Enter **next** in the command line and hit the **Enter** key. You will see the source window single step and the command appears in the Command tab, along with the response. Click the **Submit** button to repeat the command.

Now open the **History** tab. You will see the two next commands (no responses) at the bottom. Select the two next entries (click on one next entry, then <Shift>-click on the other next entry to add it),

A screenshot of the 'History' tab in the debugger interface. It shows a list of two entries, both labeled 'next', each with a small blue selection box to its left.

right-click, select **Copy**, open the Script tab, right-click and select **Paste**. In the pop-up window, enter Next for the file name and click the **Save** button. Now you have a script with the two Next entries saved as Next.ds. You can execute the script by pressing the Play  button in the Script tab.



This can be very useful in debugging when you must repeatedly execute a sequence of commands, or if you want to create canned examples for demonstration purposes.

10. We can learn a little about how the game is designed by using the command line to examine the block selection parameters. Re-enable the breakpoint in the **generateFallingBlock** function (remember? In the Breakpoints tab). Now run the game until it stops on that breakpoint.

Enter the command **set var blocknr = 2** and hit the **Enter** key (or click the **Submit** button). Now run the game again and see what shape you have selected.

Can you determine what value the square block has using this method?


11. Now let's combine what we've learned about using the command line, creating scripts and breakpoints to "fix" the game to our advantage. You should be stopped in the **generateFallingBlock** breakpoint, on line **297**.

First, create a new script file using the **set var blocknr** assignment using the value you determined for the square. Save the script in the default location with the name **favorite-block**.

Next, go to the breakpoints window. Select the breakpoint for line **297**, right-click and select **Properties**. Locate the **On break, run script** panel and use the drop-down arrow on the text box to select your **favorite-block.ds** script (Note that the pathname will be specific to your host computer). Click the **Continue Execution** checkbox so that each time the breakpoint is hit, our script will execute and then the program will automatically continue.



Click the **OK** button to save the property changes, and run the game.

You should now see only squares as the falling blocks. When you are satisfied with this, stop the program by hitting the pause  button. If the game finishes, you can simply start a new game in the X window.

12. Now you may have noticed the preview window still shows a random block, rather than our "desired" square block. Using your program analysis skills and the techniques you have learned, make additional enhancements that will do the following:
 - a. Show the square in the preview window.
 - b. Make the square shapes red.

As a hint, everything you need to do is contained inside the **generateFallingBlock** function. Give it a try!

There are many more features that can be explored in the DS-5 debugger that would take a full day class to touch upon. But in the limited time available you have had a good introduction to some of the capabilities of the software system. Much more is possible when using the DSTREAM hardware tools, but that will wait for another day.

Solution

Square block = 6

Color red = 0

To rig game:

- Make a new script with the following lines:
 - set var color_next=0
 - set var blocknr_next=6
- Set a breakpoint at line 307
- In the breakpoint properties, execute your new script and continue execution

Revision History

Date	Version	Revision
25 June 13	00	Initial Draft
17 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>