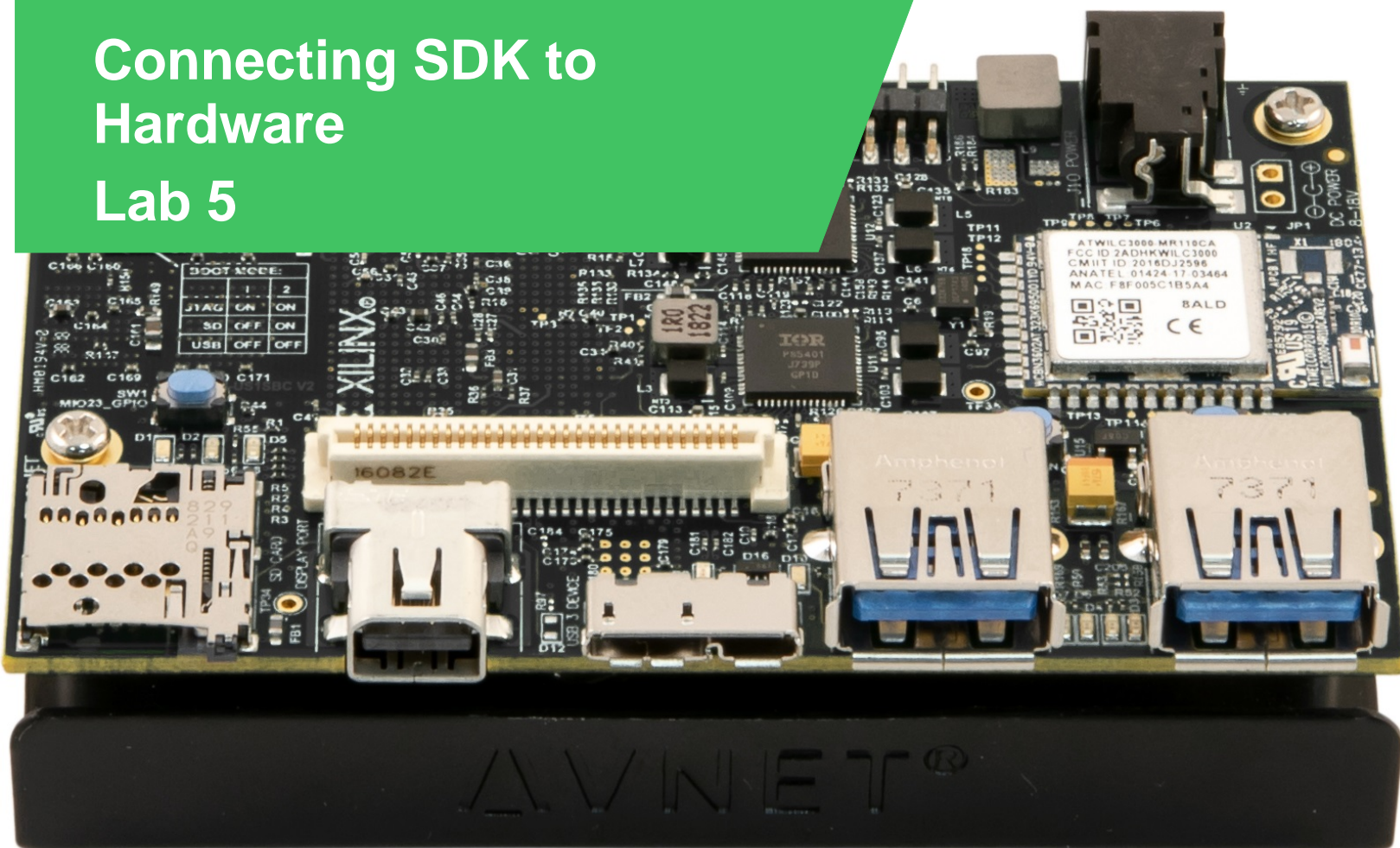


Avnet Technical Training Course

Connecting SDK to Hardware Lab 5



Tools:	2019.1
Training Version:	v12
Date:	July 2019

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Lab 5 Overview

At last we will now be able to power up the Zynq hardware. Using the example application projects and the SDK, we will connect to the hardware through a JTAG connection. The bitstream will be programmed into the PL, the ARM processor configured, and the code bootloaded to the appropriate memory. SDK can be used to simply run the application or use a full-featured application debugger to step through the code.

Lab 5 Objectives

When you have completed Lab 5, you will know how to:

- Set up the hardware for operation
- Program a bitstream to the PL
- Configure the ARM PS over JTAG using TCL
- Run an application
- Debug an application

Experiment 1: Setup Hardware and Download Bitstream

Xilinx Zynq SoCs are SRAM-based devices, meaning they are volatile. The PL is volatile and the ARM registers are also volatile. In production, the PL hardware definition (or bitstream) and the ARM PS configuration code (or First-Stage Boot Loader FSBL) are stored in Flash on the board, such as a Quad-SPI Flash or SD Card.

Before storing a hardware/software design to Flash, developers will want to experiment with the code – both PL hardware code as well as software application code. In this experiment, we will focus on software application code experimentation directly from SDK. Making use of a JTAG connection to the hardware, SDK will be able to perform all the functions that would normally be performed from Flash.

The SDK-SoC JTAG interface may be used to accomplish all these things:

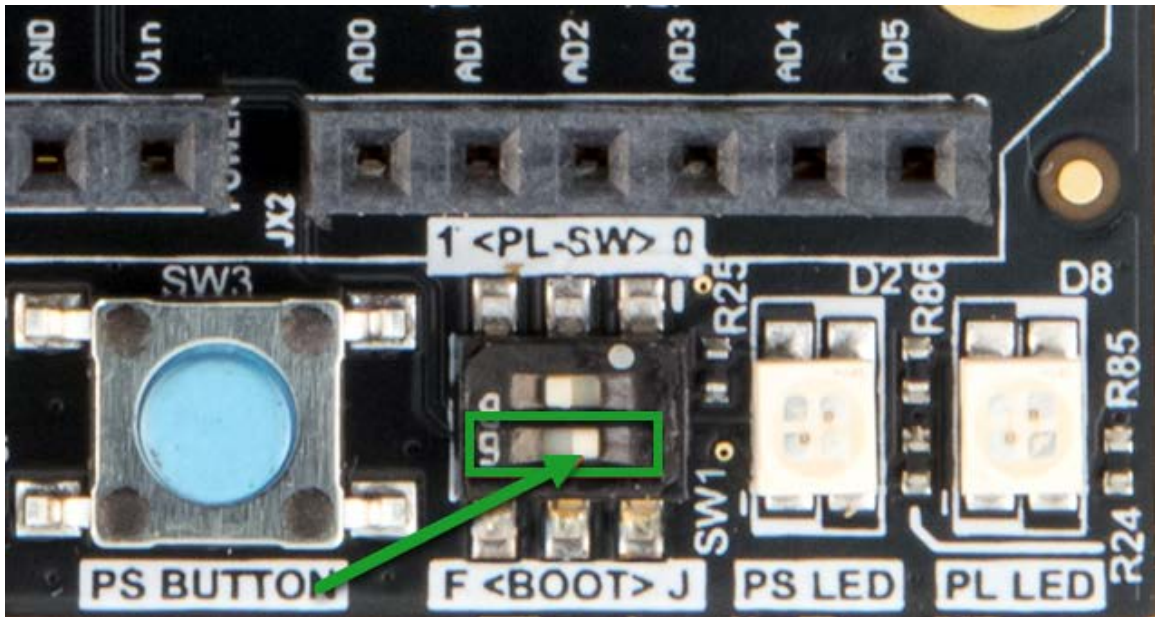
- Read and write ARM registers
- Configure the PL with a bitstream
- Program attached QSPI Flash
- Upload application code to on-chip RAM or DDR3L
- Application debug

Experiment 1 General Instruction:

Setup the hardware and connect all cables. Determine the COM Port assignment on your PC. Configure the SoC PL with a bitstream.

Experiment 1 Step-by-Step Instructions:

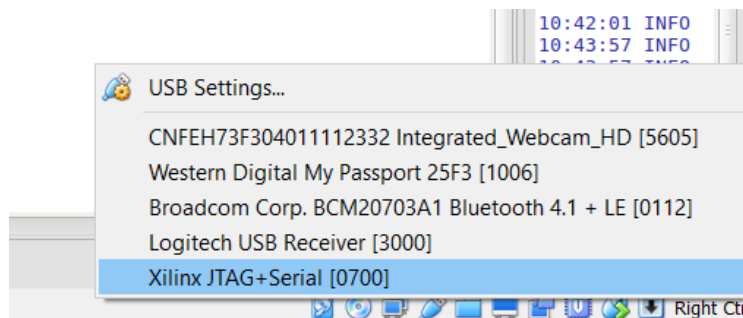
1. Now we are going to set up our hardware. Set Boot Mode jumpers to **Cascaded JTAG Mode**:
 - a. MiniZed – Set outside dip switch towards J (JTag Boot)



MiniZed

Figure 1 –MiniZed Boot Selection


2. Connect the MiniZed USB-JTAG/UART port J2 to your PC, make the JTAG and serial port available to the Virtual Machine by right clicking on the USB Icon and selecting Xilinx JTAG + Serial option



© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Figure 2 – MiniZed USB-JTAG/UART Installed Correctly

3. Open Terminal, such as **GTK Term**, and set the **COM port** to active COM setting for your board and set the **Baud Rate at 115,200**.
4. In SDK, select **Xilinx Tools → Program FPGA** or click the  icon.
5. SDK will already know the correct .bit file (and .bmm if your future hardware platform includes that) since this was imported with the hardware platform. Click **Program**. When the DONE LED lights blue (D3 on MiniZed), the PL has configured successfully. Look for the message “FPGA configured successfully with bitstream” in the SDK log.

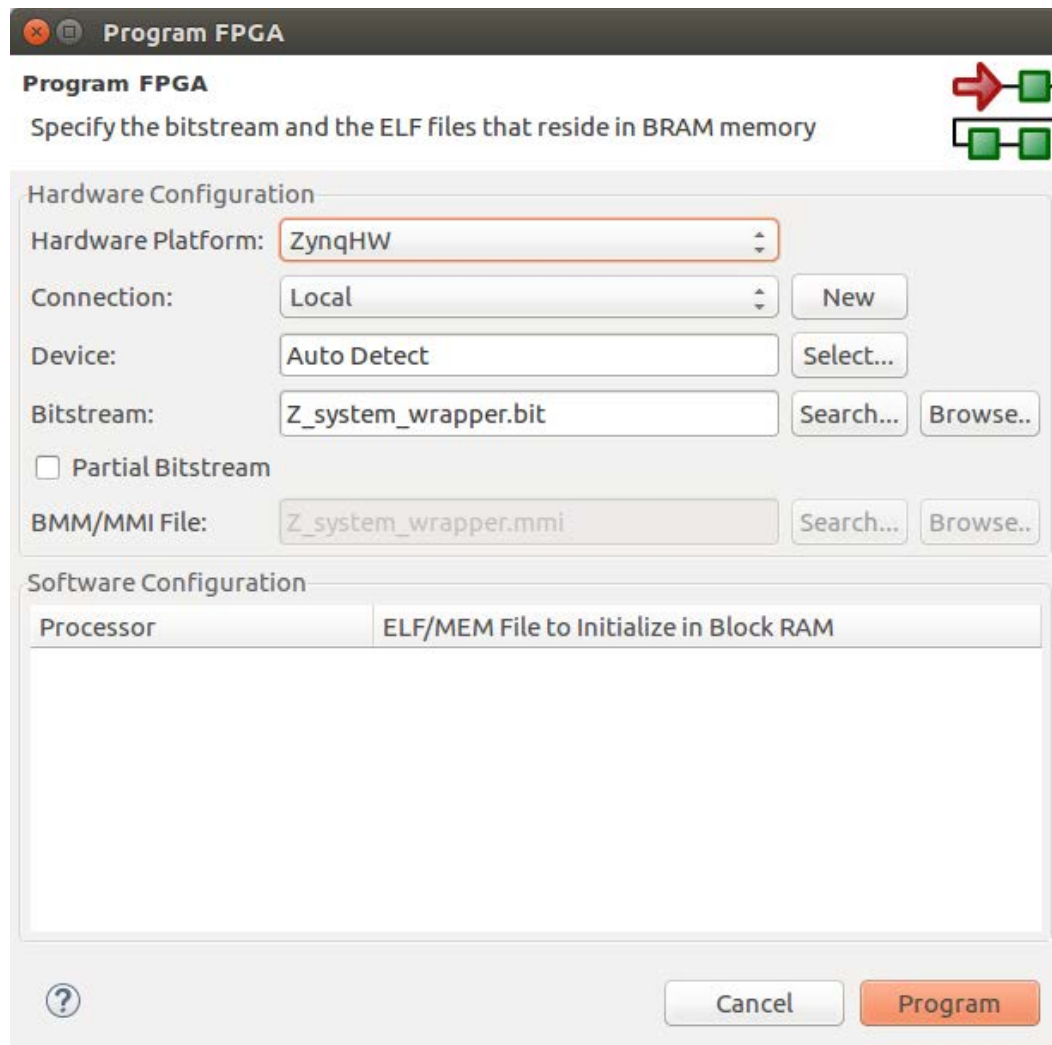


Figure 3 – Program FPGA

Questions:

Answer the following questions:

- *What can the JTAG interface be used for?*

1. _____

2. _____

3. _____

4. _____

- *Under what conditions must the hardware platform first be downloaded into the PL?*

Experiment 2: Running an Application


SDK is now ready to run or debug an application. In this experiment, the previous Hello World and Memory Tests will be run.

Experiment 2 General Instruction:

Run the Hello_Zynq and Test_Memory applications on the hardware, viewing the results on the stdout Terminal. Run the edited Test_Memory to see the expanded test window.

Experiment 2 Step-by-Step Instructions:

Since we previously setup the hardware and configured the hardware platform to FPGA, the blue DONE LED should be lit, and the hardware should be ready to accept an application to run.

1. Right-click on the Hello_Zynq application and select **Run As → Run Configurations...**
2. Select **Xilinx C/C++ Application (System Debugger)** and then click the 'New' icon . You may also simply double click **Xilinx C/C++ Application (System Debugger)**.

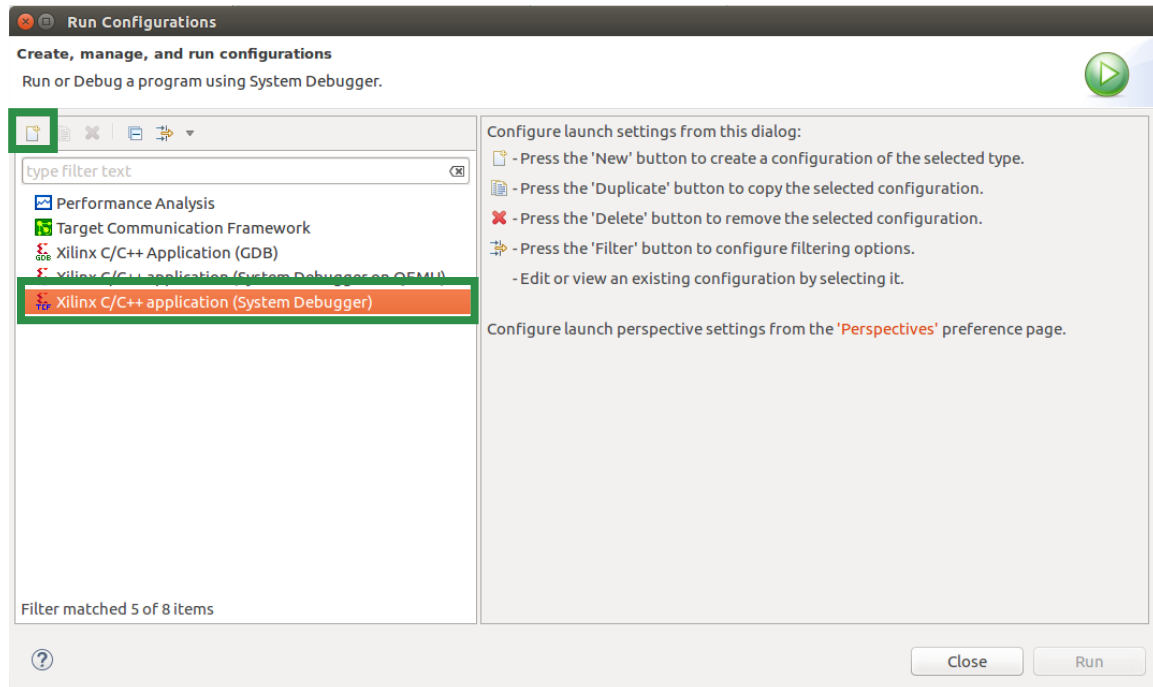


Figure 4 – Create a New Xilinx C/C++ Application Run Configuration

SDK creates the new Run Configuration and automatically assigns a name to the configuration `<application_name> <active_configuration>`, which in this case is *Hello_Zynq Debug*.

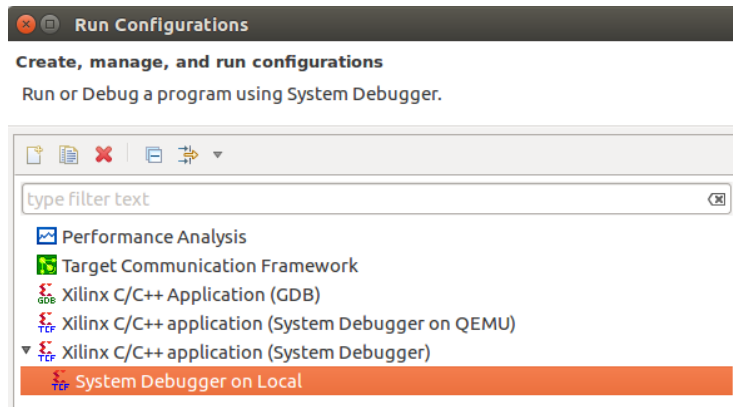


Figure 5 – New Run Configuration

Recall that the example code copied over for the Hello_Zynq application was quite simple. Other than the UART statements, there wasn't much else. There was nothing in Hello_Zynq to set up the ARM registers for the designed clocking, peripherals, and I/O connections. If the Hello_Zynq application doesn't configure the ARM processing system, then how will it get done?

1. Ensure the connection type is set to **Standalone Application Debug**
- 2.

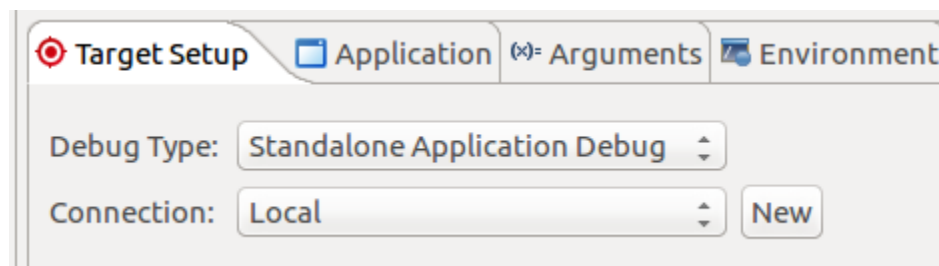


Figure 6 – Setting the connection type

3. Note that by default, the ps7_init.tcl will be sourced prior to running your application. That is how the ARM gets initialized for proper operation. Recall that this ps7_init.tcl was one of the outputs reviewed in Lab 1.



Figure 7 – ps7_init.tcl Script to Source

4. Make sure both Run ps7_init and Run ps7_post_config are enabled

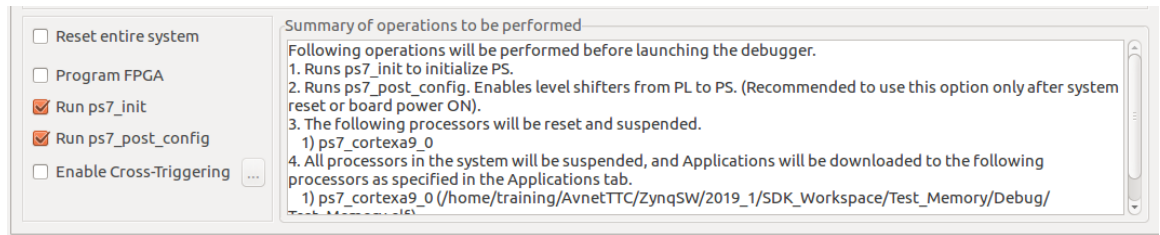


Figure 8 – Ensuring Init and Post Config Scripts are Run

5. Ensure the application is set to download your desired application

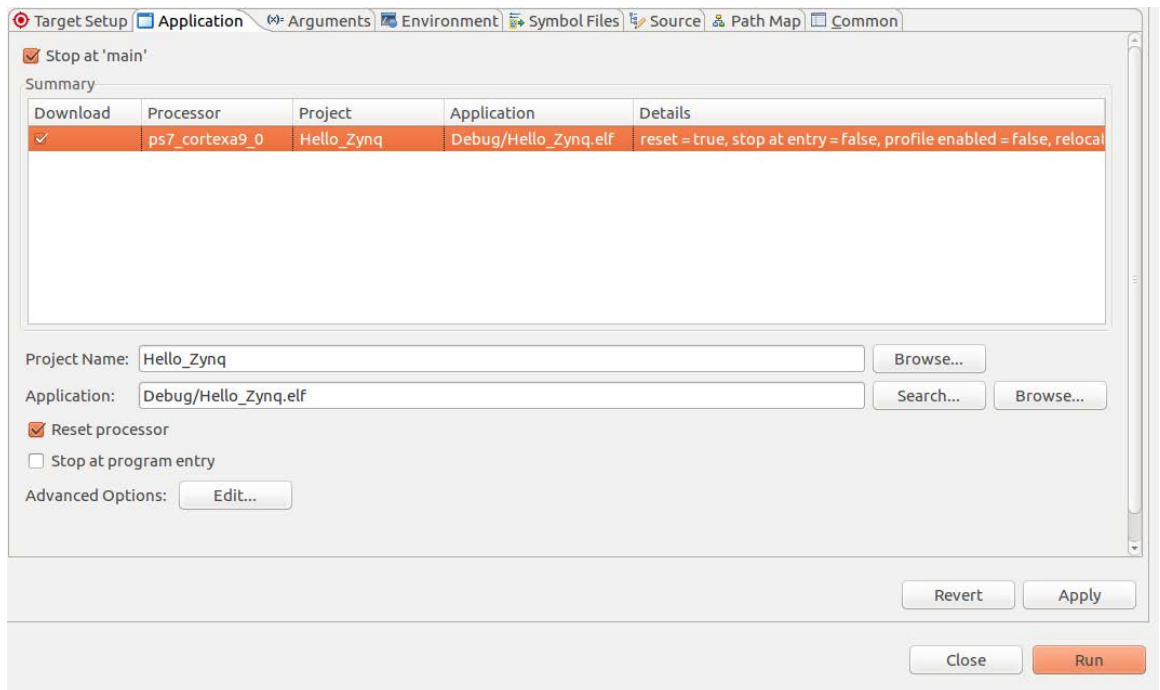


Figure 9 – Setting the Application to download

6. Return back to SDK and Click **Run**.

You can monitor the progress of the download in the lower right-hand corner.

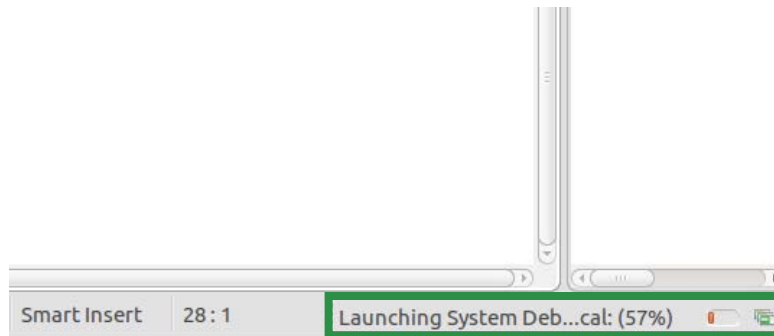


Figure 10 – Download Progress

SDK will download the Hello_Zynq ELF to the on-chip RAM (because this is what we set in the linker script in Lab 4) and begin executing the code. You can see the output in Tera Term.

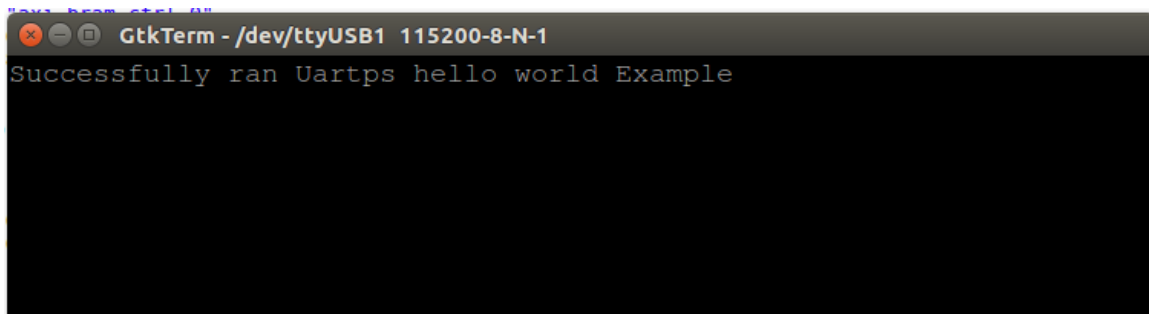
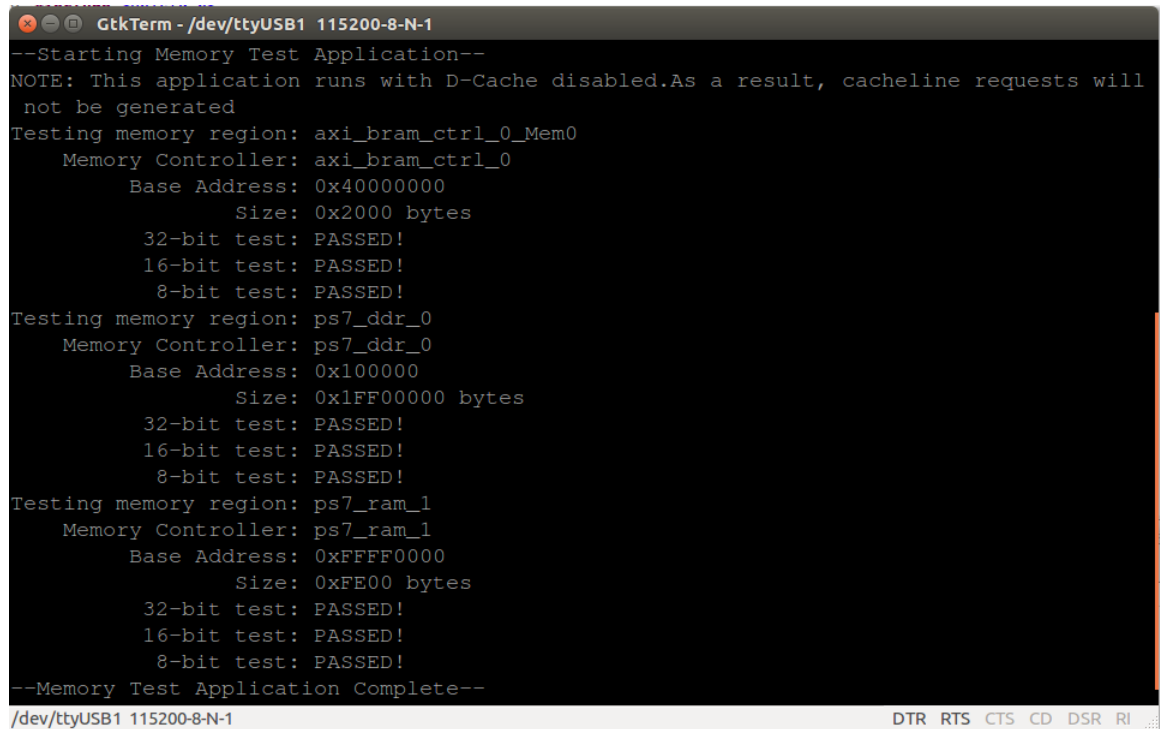


Figure 11 – Hello World

7. Repeat steps 1 through 4 for application Test_Memory, including setting up a new Run Configuration, confirming the Device Initialization, and setting up the Tera Term

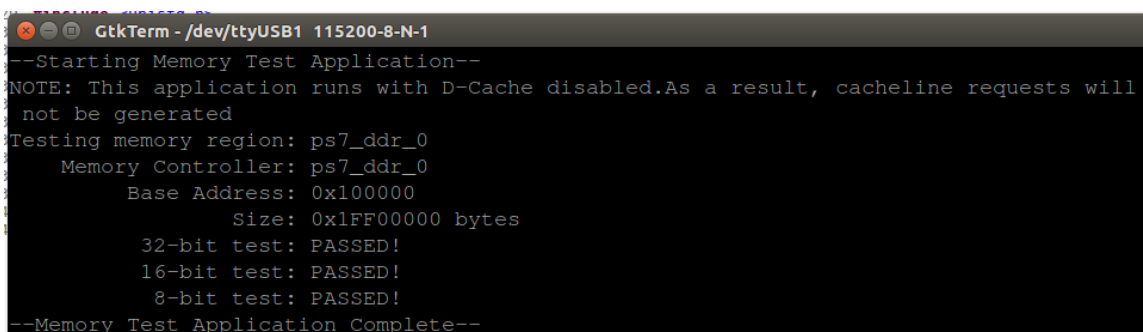
Connection. This test completes very quickly as it only tests the first 4 KB in the three tested memory spaces. Click **Yes** to the Conflict, abort existing launch configuration screen.



```
GtkTerm - /dev/ttyUSB1 115200-8-N-1
--Starting Memory Test Application--
NOTE: This application runs with D-Cache disabled.As a result, cacheline requests will
not be generated
Testing memory region: axi_bram_ctrl_0_Mem0
  Memory Controller: axi_bram_ctrl_0
    Base Address: 0x40000000
    Size: 0x2000 bytes
    32-bit test: PASSED!
    16-bit test: PASSED!
    8-bit test: PASSED!
Testing memory region: ps7_dds_0
  Memory Controller: ps7_dds_0
    Base Address: 0x100000
    Size: 0x1FF00000 bytes
    32-bit test: PASSED!
    16-bit test: PASSED!
    8-bit test: PASSED!
Testing memory region: ps7_ram_1
  Memory Controller: ps7_ram_1
    Base Address: 0xFFFF0000
    Size: 0xFE00 bytes
    32-bit test: PASSED!
    16-bit test: PASSED!
    8-bit test: PASSED!
--Memory Test Application Complete--
/dev/ttyUSB1 115200-8-N-1 DTR RTS CTS CD DSR RI
```

Figure 12 – Memory Tests Complete

8. Repeat the run again using the previously edited application Test_Memory_FullDDR to see if you successfully edited the Memory Test code to test 1MB of DDR3L. Click **Yes** to the Conflict, abort existing launch configuration screen.



```
GtkTerm - /dev/ttyUSB1 115200-8-N-1
--Starting Memory Test Application--
NOTE: This application runs with D-Cache disabled.As a result, cacheline requests will
not be generated
Testing memory region: ps7_dds_0
  Memory Controller: ps7_dds_0
    Base Address: 0x100000
    Size: 0x1FF00000 bytes
    32-bit test: PASSED!
    16-bit test: PASSED!
    8-bit test: PASSED!
--Memory Test Application Complete--
```

Figure 13 – 1MB of DDR3L Tested

The same test runs as before. However, this time it takes slightly longer since a larger space is being tested.

9. Close GTK Term

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Questions:

Answer the following questions:

- *How does the ARM get initialized when running an application from SDK over the JTAG connection?*

- *How much memory is tested by default?*

Experiment 3: Debug an Application

Debugging applications is a very important part of the software development process. During this experiment, we will exercise the debugger with the Peripheral Test application.

Xilinx utilizes the SDK System Debugger, based on the Target Communications Framework (TCF). According to Xilinx, System Debugger delivers true multi-processor SoC design and debug. For example, in a Zynq-based design, System Debugger can display both ARM CPUs and multiple PL soft-processors, in the same debug session, through a single JTAG cable for an unprecedented level of insight between the hardened processing system and any additional processing that you've added to the programmable logic.

- Based on the Target Communication Framework (TCF)
- Homogenous and heterogeneous multi-processor support
- Linux application debug on the target
- Hierarchical Profiling
- Bare-metal and Linux development
- Supporting both SMP and AMP designs
- Associate hardware and software breakpoints per core
- NEON™ library support

The traditional GDB Debugger continues to work with Zynq applications as well.

Experiment 3 General Instruction:

Launch the Peripheral Test application in the Debugger. Step through a few lines of code. Set breakpoints. Observe memory.

Experiment 3 Step-by-Step Instructions:

1. Right-click on the *Test_Peripherals* application and choose **Debug As** to view the options. Notice that we have two *Launch on Hardware* options – GDB and System Debugger. Don't select either of these now. Rather select **Debug Configurations...**

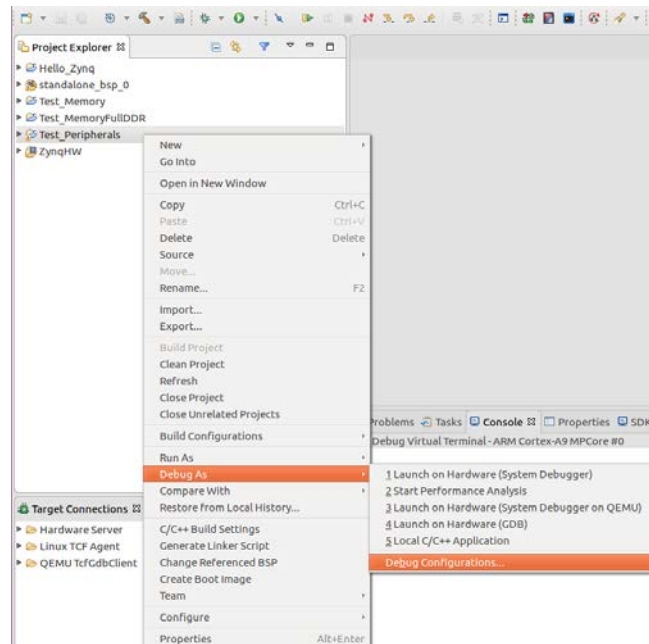



Figure 14 – Debug As Option

2. Notice that the Configurations menu remembers the previous Run Configurations from the previous experiments. Select **Xilinx C/C++ application (System Debugger)** then click the new  icon.

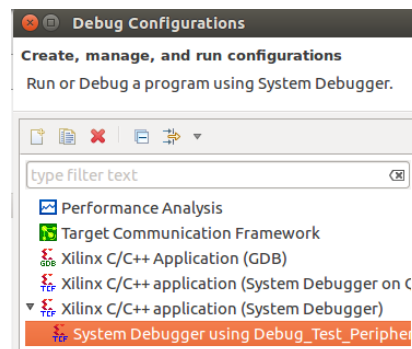


Figure 15 – Create New System Debugger Configuration

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

- If you hadn't previously programmed the PL, you could check the **Reset entire system** and **Program FPGA** checkboxes. However, we have already programmed the PL (Blue Led D3 is illuminated) so we will leave these unchecked. If D3 is not illuminated check the Reset entire system and Program FPGA checkboxes.

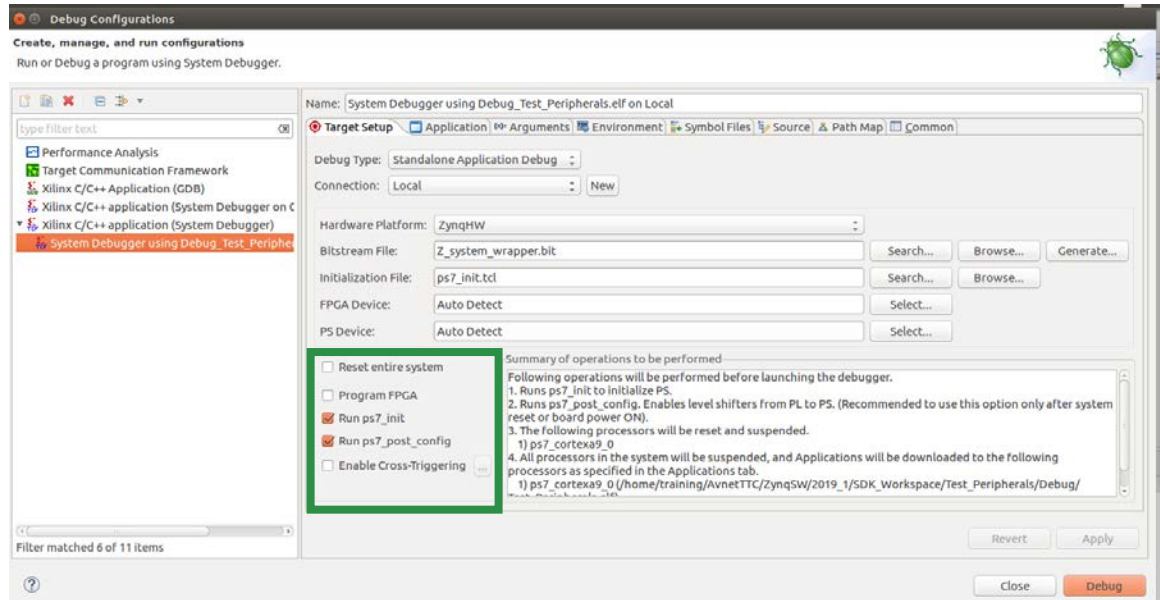


Figure 16 – Target Setup

- Feel free to explore the other tabs, but we will not make any other changes. Click **Debug**.
- The code will be downloaded to the target memory, which in this case is DDR3L. However, the application will not be run. Instead, the SDK's windows will be re-configured to show you what is known as the "debugging perspective." Click **Yes** to confirm the perspective switch.

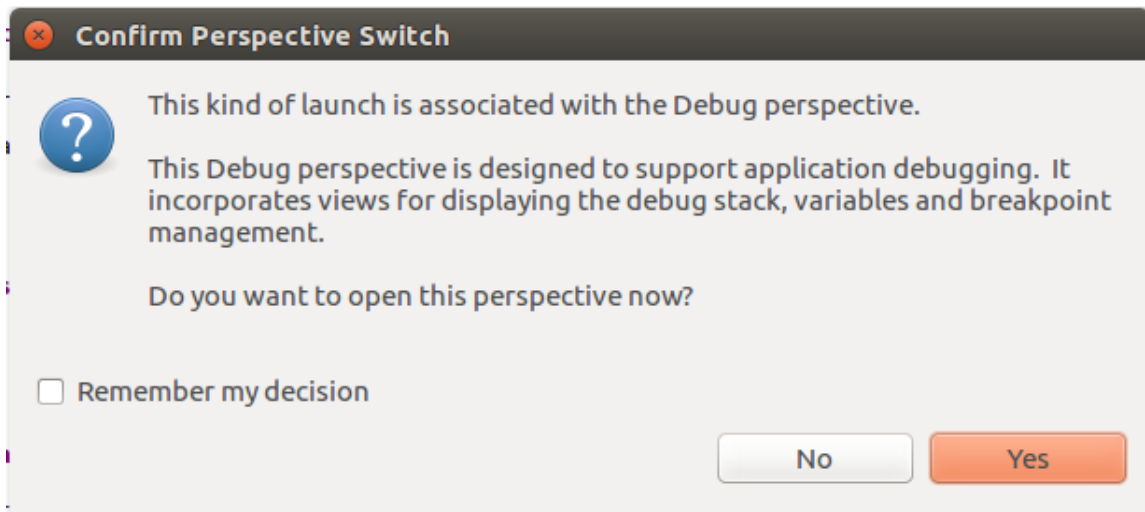


Figure 17 – Confirm Perspective Switch

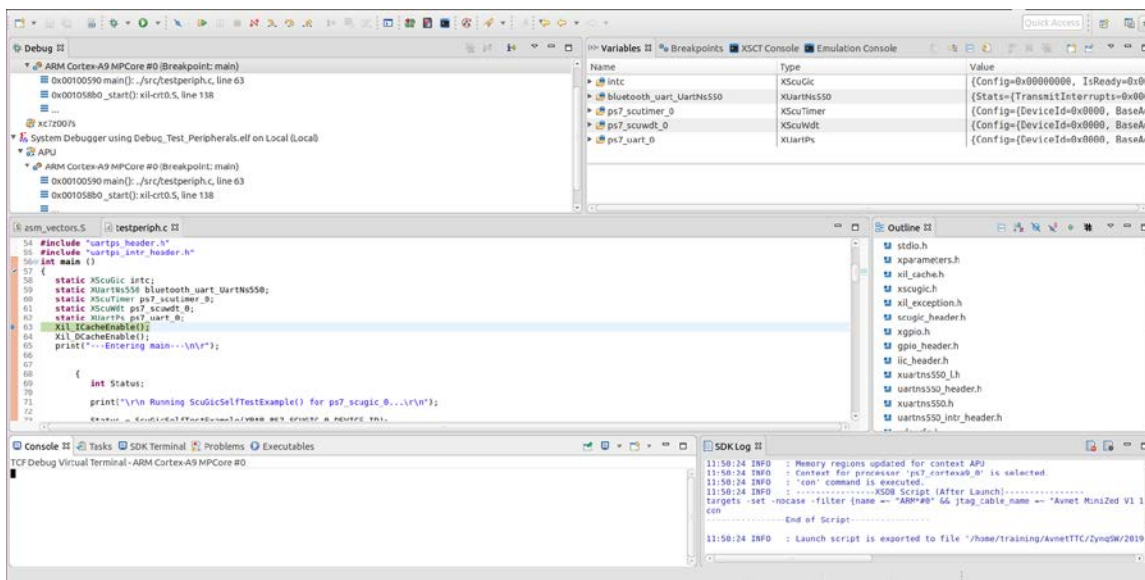


Figure 18 – Debugging Perspective

It is important to note that the Zynq processor is now in a halted state; therefore the application is not being executed. In the code editing window, you will now see a green bar, which shows the line of code which will be executed next.

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.


NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

```

63 Xil_ICacheEnable();
64 Xil_DCacheEnable();
65 print("---Entering main---\n\r");
66

```

Figure 19– Debug Waiting for User Input

6. *****Make Sure GTK Term is Closed*****At the bottom of the SDK screen, select the *SDK Terminal* tab. Click the green “Connect to serial port” icon . Change the *Settings* for 115200 baud, 8 data bits, 1 stop bit, no parity and no flow control. Choose the COM port identified previously. Click **OK**.

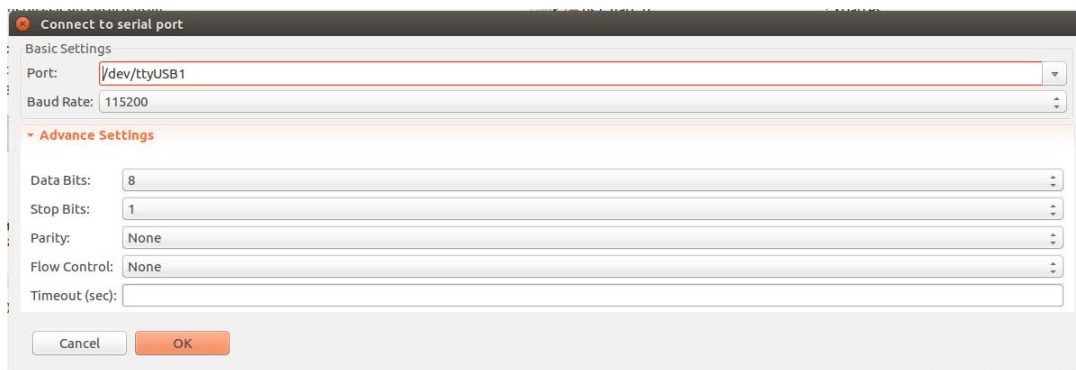


Figure 20 – Terminal Settings

7. In the header of the Debug pane at the top of the screen, you should see a number of icons. We shall use the “Step Over” icon to advance the application by one line of code. Click this icon one time and observe the effect on the code editing window and the UART Terminal.



As you can see, the processor has executed the first line of code. The green bar indicates the next line of code waiting to be executed.

Each time the green bar is shown in the code window, the Zynq processor has advanced but then stopped executing again. Next, we'll set a breakpoint. For ease of explanation, it would be useful to have the editor show line numbers.

8. If line numbers are not already shown, select **Window → Preferences**. Within the *Preferences* dialog, browse to **General → Editors → Text Editors**. Click the checkbox for **Show line numbers**. Click **OK**.

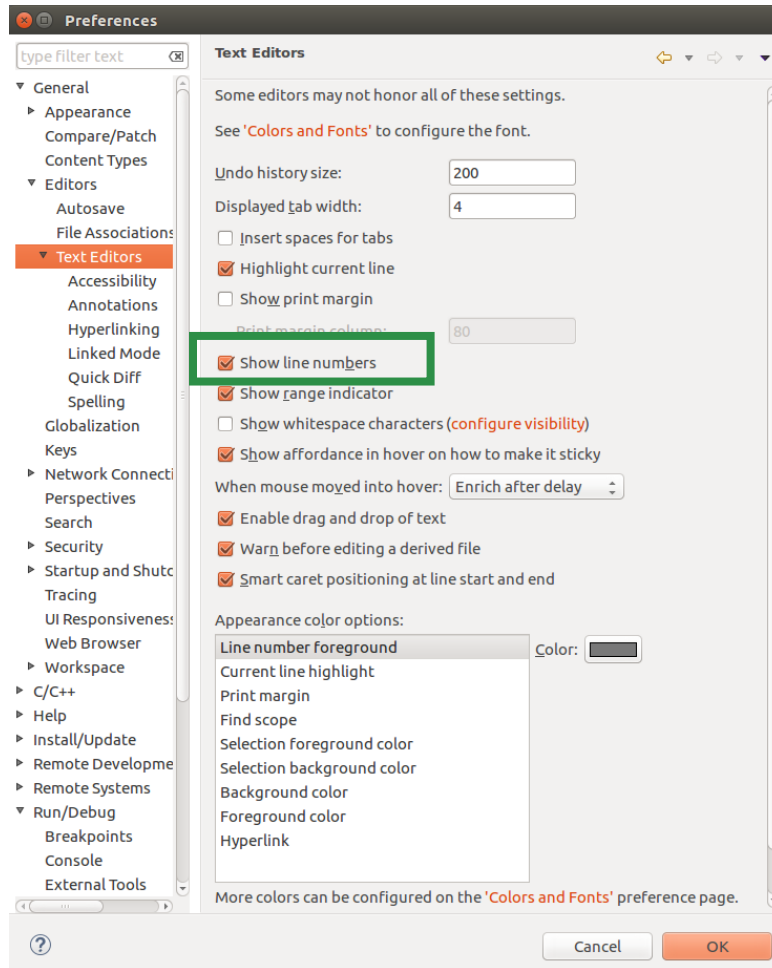


Figure 21 – Show Line Numbers in the Text Editor

9. Browse to Line 273 in testperiph.c, where the code is beginning the SCU timer. Set a breakpoint on Line 273 by double-clicking in the blue column to the left of the line number. A small “circle with a checkmark” should appear next to the line.

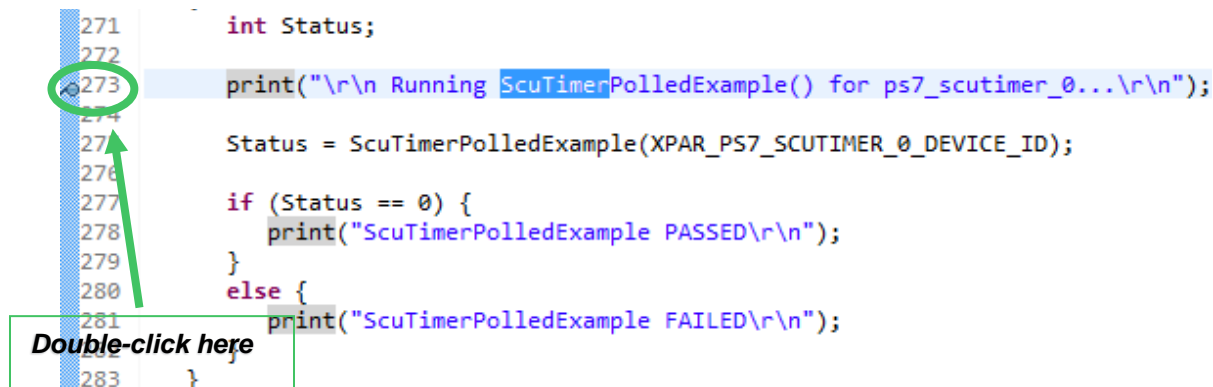


Figure 22 – Setting a Breakpoint

10. In the upper right hand area of the perspective, select the *Breakpoints* tab. This area allows you to disable or delete breakpoints. Breakpoints can also be deleted by double-clicking again to the left of the line number.

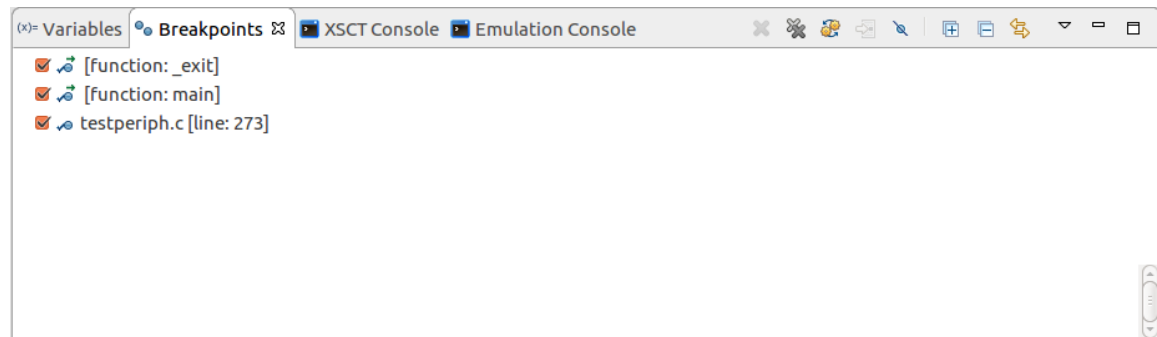


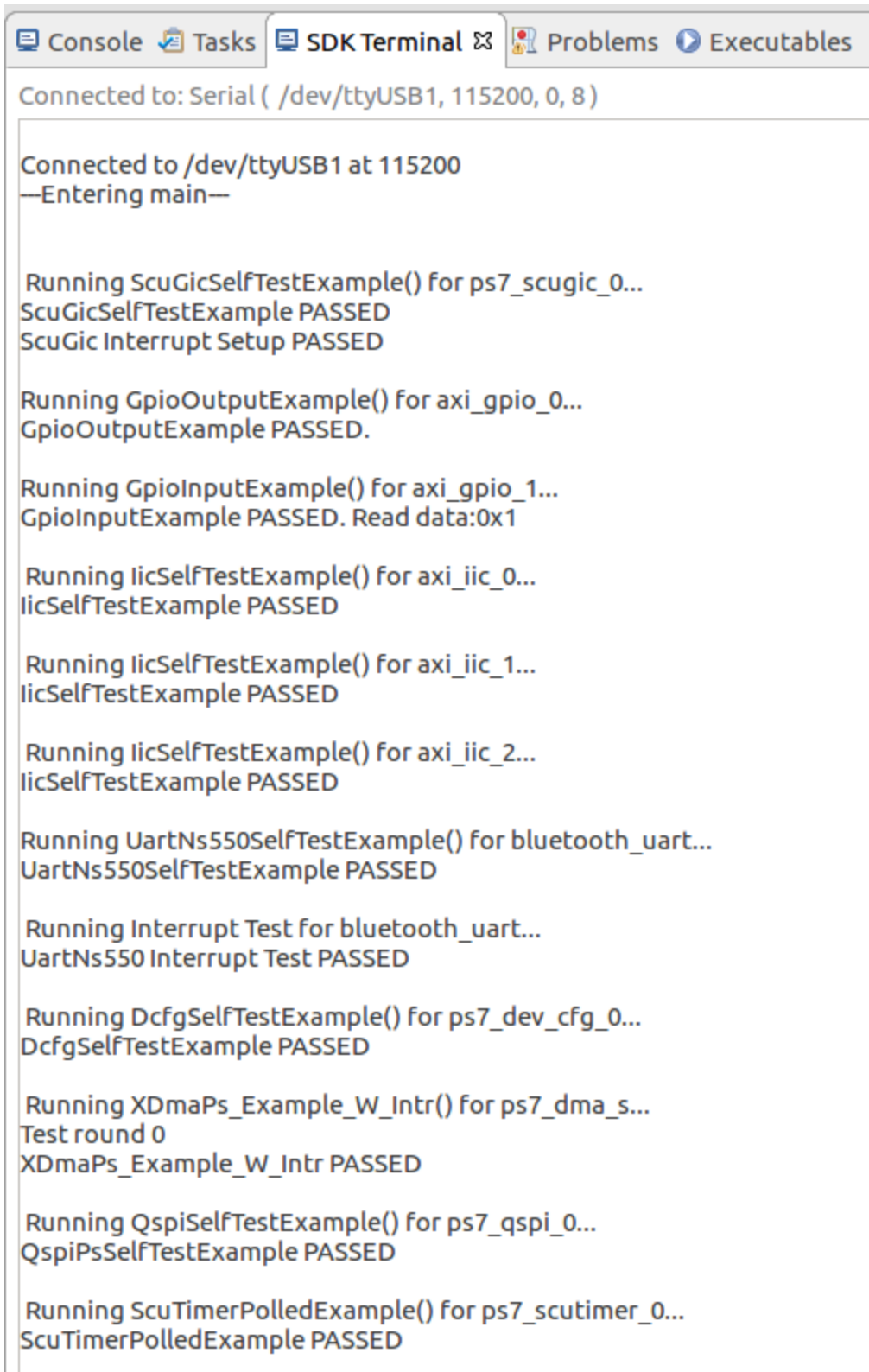
Figure 23 – Breakpoints

11. Click the “Resume” icon which is depicted by the Green arrow.



12. A few tests have now passed, as indicated by the results in the *SDK Terminal* window. The next sequence of code will test the QSPI. Click the Resume icon again.

Since this application is not in a loop, the application has finished. The complete results from the Peripheral Test application are shown below.



```
Console Tasks SDK Terminal Problems Executables
Connected to: Serial ( /dev/ttyUSB1, 115200, 0, 8 )

Connected to /dev/ttyUSB1 at 115200
—Entering main—

Running ScuGicSelfTestExample() for ps7_scugic_0...
ScuGicSelfTestExample PASSED
ScuGic Interrupt Setup PASSED

Running GpioOutputExample() for axi_gpio_0...
GpioOutputExample PASSED.

Running GpioInputExample() for axi_gpio_1...
GpioInputExample PASSED. Read data:0x1

Running IicSelfTestExample() for axi_iic_0...
IicSelfTestExample PASSED

Running IicSelfTestExample() for axi_iic_1...
IicSelfTestExample PASSED

Running IicSelfTestExample() for axi_iic_2...
IicSelfTestExample PASSED

Running UartNs550SelfTestExample() for bluetooth_uart...
UartNs550SelfTestExample PASSED

Running Interrupt Test for bluetooth_uart...
UartNs550 Interrupt Test PASSED

Running DcfgSelfTestExample() for ps7_dev_cfg_0...
DcfgSelfTestExample PASSED

Running XDmaPs_Example_W_Intr() for ps7_dma_s...
Test round 0
XDmaPs_Example_W_Intr PASSED

Running QspiSelfTestExample() for ps7_qspi_0...
QspiPsSelfTestExample PASSED

Running ScuTimerPolledExample() for ps7_scutimer_0...
ScuTimerPolledExample PASSED
```

Figure 24 – Complete Peripheral Test (MiniZed)

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

13. Under the *Debug* tab in the upper left-hand corner, right-click on **System Debugger using Debug_Test_Peripherals.elf on local (Local)** and select **Relaunch**. Click **OK**.

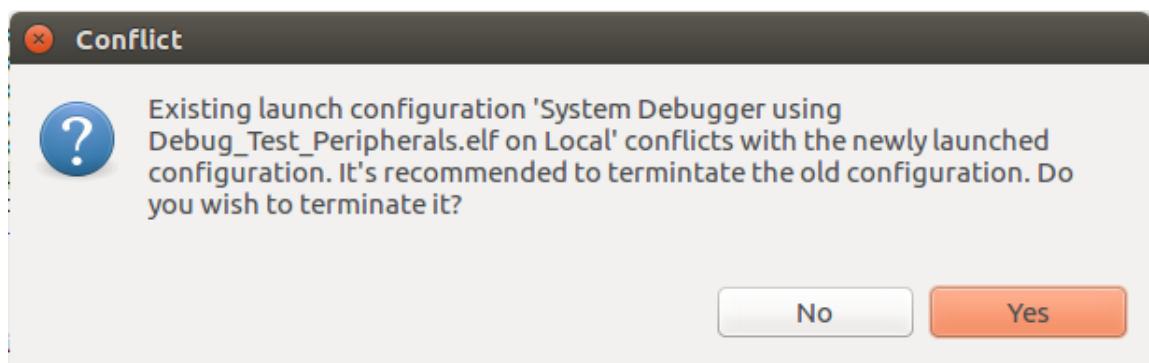


Figure 25 – Relaunch Test_Peripherals Debug

14. Use the *Breakpoints* tab to disable the breakpoint at Line 273. Simply uncheck the box.
15. This time, click Resume, then immediately click the Suspend icon, which is the yellow "pause" icon.



16. If you end up suspended inside one of the other functions, the SDK debugger will open the function with the green bar showing exactly where you are paused. Click the Step Return icon to get back to the testperiph.c. Depending on where you suspended execution, you may have to click the Step Return several times. Click Resume to continue the test.



17. Click the Disconnect icon to end the debugging session.



18. Finally, in the top-right corner of the screen, click the “C/C++” icon to return the SDK to the code editing perspective. Alternatively, to close the Debug perspective, right-click on Debug and select **Close**.



Exploring Further

If you have more time and would like to investigate further...

- Debug the Test_Memory application. Make use of the following debugger features from the **Window → Show View** menu.
 - **Memory**
 - **Variables**
 - **Registers**
- The first Test_Memory application targets the application code at RAM_0. Then, the application also runs a destructive memory test on RAM_0. How is that possible? Can you prove it?

This concludes Lab 5.

Revision History

Date	Version	Revision
12 Nov 13	01	Initial release
23 Nov 13	02	Revisions after pilot
01 May 14	03	ZedBoard.org Training Course Release
30 Oct 14	04	Revised for Vivado 2014.3
31 Dec 14	05	Revised for Vivado 2014.4
09 Mar 15	06	Merge MicroZed and ZedBoard instructions
18 Mar 15	07	Finalize SDK 2014.4
Oct 15	08	Updated to SDK 2015.2
Aug 16	09	Updated to SDK 2016.2
Jun 17	10	Updated to 2017.1 for MiniZed + Rebranding
Jan 18	11	Updated to Vivado/SDK 2017.4
July 19	12	Updated to Vivado/SDK 2019.1

Resources

www.minized.org

www.microzed.org

www.picozed.org

www.zedboard.org

www.xilinx.com/zyng

www.xilinx.com/sdk

www.xilinx.com/vivado

www.xilinx.com/support/documentation/sw_manuals/ug949-vivado-design-methodology.pdf

www.xilinx.com/support/documentation/sw_manuals/ug1046-ultrafast-design-methodology-guide.pdf

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Answers

Experiment 1

- *What can the JTAG interface be used for?*
 1. Read and write ARM registers
 2. Configure the PL with a bitstream
 3. Program attached QSPI Flash
 4. Upload application code to on-chip RAM or DDR3L
 5. Application debug
 6. Performance analysis and profiling
- *Under what conditions must the hardware platform first be downloaded into the PL?*

If the application only uses the Zynq PS, then programming the PL is not necessary. If the application makes use of something in the PL, then the PL must be programmed first. Unlike an off-the-shelf microprocessor with all built-in peripherals, the Zynq SoC starts out with the PL as a blank hardware device. The hardware identity, or bitstream, must be first downloaded to the PL. Any PL peripherals or co-processors don't exist in the PL until after the hardware platform bitstream is configured to the PL.

Experiment 2

- *How does the ARM get initialized when running an application from SDK?*

ps7_init.tcl that was provided with the hardware platform

- *How much memory is tested by default?*

4 KB

Troubleshooting

Any Windows Security Alerts will cause issues. Click Allow Access, reboot, then retry. Hopefully this will not affect anyone since we are covering this in Lab 0.

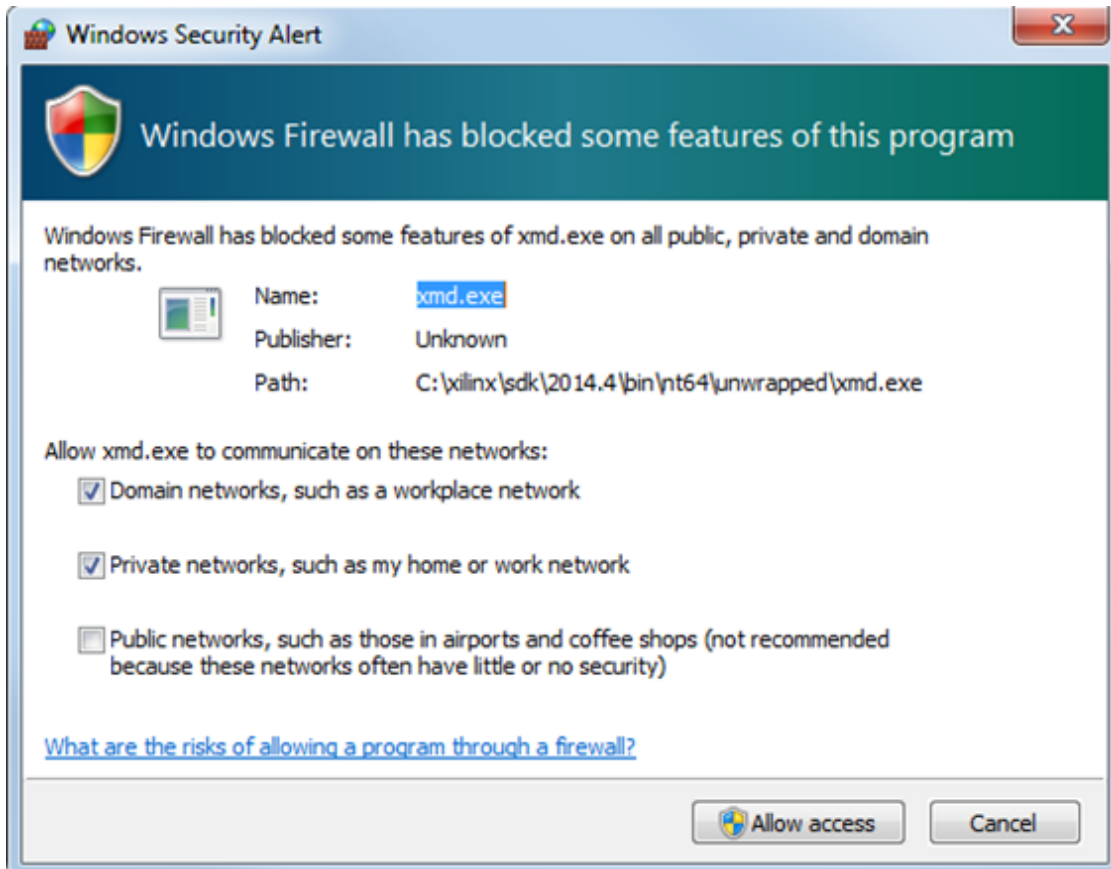


Figure 26 – Windows Security Alert