Implementing Linux on the Zynq™-7000 SoC

Lab 4.1

Using SDK for Linux Application Development

September 2012
Version 05

Table of Contents

Lab 4.1 Overview

This lab is a short demonstration of the Linux environment on the Zynq™ EPP, illustrating how to develop a Linux software application.

In this lab, you will use the Xilinx Software Development Kit (SDK) to create a simple Linux software application project. The target ZedBoard will automatically boot Linux from an SD card with the Linux kernel as was configured in the previous lab exercises.

Using SDK, you will create a new workspace, import a hardware platform and build a Linux application.

Lab 4.1 Objectives

When you have completed Lab 4.1, you will know how to do the following:

- Create an SDK software application workspace
- Import an existing Zynq EPP hardware platform
- Create and build a Linux software application project
- Open and use the Remote System Explorer (RSE) tool environment
- Download and run an application on development hardware
- Explore Linux operations on the Zynq EPP

Experiment 1: Basic Application Development Workflow

This lab comprises three primary steps: You will create an SDK software workspace, add the "Hello World" software application to the workspace, and execute the software application on ZedBoard.

SDK Workspace

The first step in this lab is to create an SDK software workspace.

The working directory of the SDK workspace for this lab is the **C:\Speedway\Fall_12\Zynq_Linux\lab4_1\workspace_linux_zed\** folder.

You will create the SDK workspace and software platform for the hardware in this directory.

Note that once SDK is launched and a workspace is set up in this directory, you will not be able to move these directories.

Adding a Software Application

By adding a new software project, SDK will automatically build and produce an executable and load format (ELF) file.

Executing a Software Application

The same new Linux application created in SDK can be loaded and executed on the ZedBoard target hardware.

Experiment 1 General Instruction:

Launch Xilinx Software Development Kit (SDK) and create a new SDK workspace in the Lab 4.1 directory. Use SDK new project template to create a new Linux application in the workspace and execute the new application code on the ZedBoard hardware.

Experiment 1 Step-by-Step Instructions:

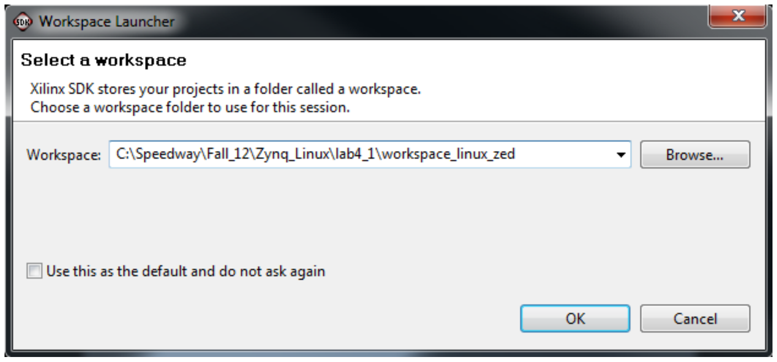
1. Launch Xilinx Software Development Kit (SDK) if not already open. **Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.2 → EDK → Xilinx Software Development Kit**.

   Figure 1 – The SDK Application Icon

   SDK creates a workspace environment consisting of project files, tool settings, and your software application. Once set, you cannot change the location of this workspace. If it is necessary to move a software application to another location or computer, use the Import and Export facilities built into SDK. A good location for the software workspace is the root directory of your PlanAhead™ or ISE® tool project. In this (and other) software only lab, neither of those other Xilinx tools are used, so workspace location is always at the discretion of the programmer.

2. Set or switch the workspace to the following folder and then click the **OK** button:

   C:\Speedway\Fall_12\Zynq_Linux\lab4_1\workspace_linux_zed\

   Select a workspace

   Xilinx SDK stores your projects in a folder called a workspace.
   Choose a workspace folder to use for this session.

   Workspace: C:\Speedway\Fall_12\Zynq_Linux\lab4_1\workspace_linux_zed

   ☐ Use this as the default and do not ask again

   Figure 2 – Switching to the Appropriate SDK Workspace

3. Close the Welcome screen if it appears in the SDK window by clicking on the **X** control in the tab.

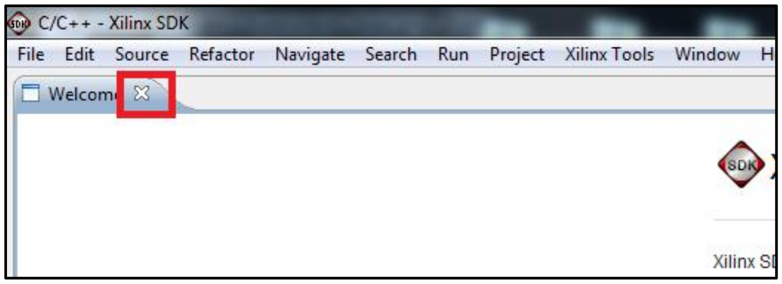

C/C++ - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window H
Welcom
SDK
Xilinx SI

Figure 3 – Closing the SDK Welcome screen

4. SDK must associate software applications with a hardware system that has been previously exported. It needs hardware configuration information so that an appropriate software platform or board support package can be built.

   Create the hardware platform specification by selecting the **File→New→Xilinx Hardware Platform Specification** menu item.

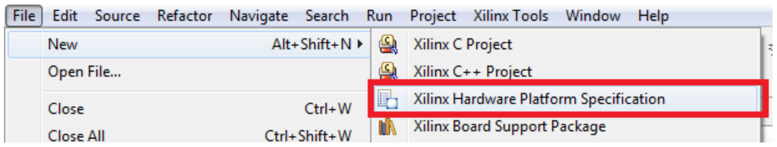

File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Xilinx C Project
Xilinx C++ Project
Xilinx Hardware Platform Specification
Xilinx Board Support Package

Figure 4 – Beginning the Creation of a Hardware Platform Specification

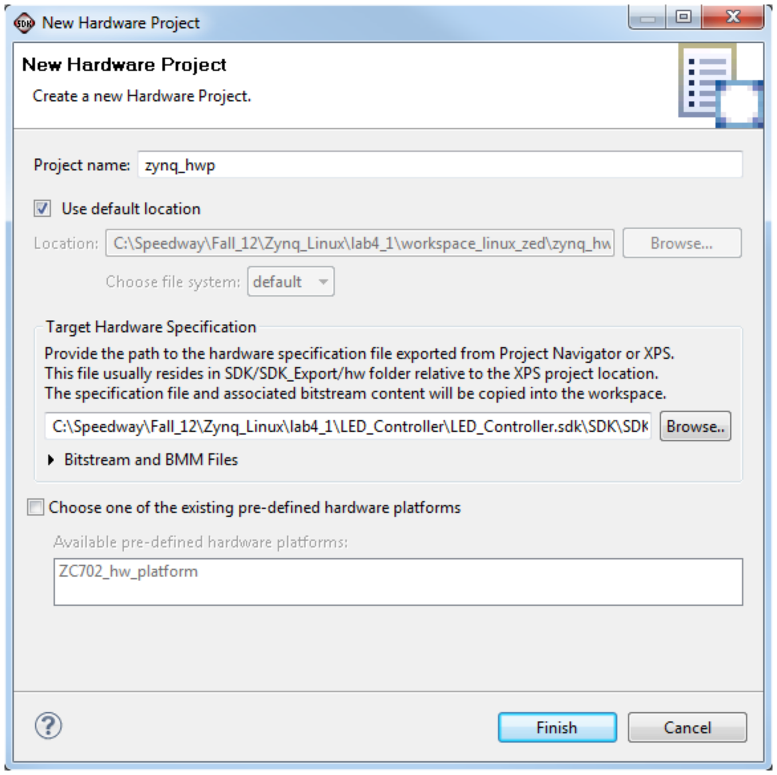
5. In the **Project name** field, use **zynq_hwp** for the name entry.

   In the **Target Hardware Specification** area, click the **Browse** button and locate the following folder:

   C:\Speedway\Fall_12\Zynq_Linux\lab4_1\LED_Controller\LED_Controller.sdk\SDK\SDK_Export\hw\

   Select the **system.xml** hardware project and click the **Open** button.

   Click the **Finish** button to create the hardware project.



New Hardware Project

Create a new Hardware Project.

Project name: zynq_hwp

☑ Use default location

Location: C:\Speedway\Fall_12\Zynq_Linux\lab4_1\workspace_linux_zed\zynq_hw

Choose file system: default

Target Hardware Specification

Provide the path to the hardware specification file exported from Project Navigator or XPS.
This file usually resides in SDK/SDK_Export/hw folder relative to the XPS project location.
The specification file and associated bitstream content will be copied into the workspace.

C:\Speedway\Fall_12\Zynq_Linux\lab4_1\LED_Controller\LED_Controller.sdk\SDK\SDK

Bitstream and BMM Files

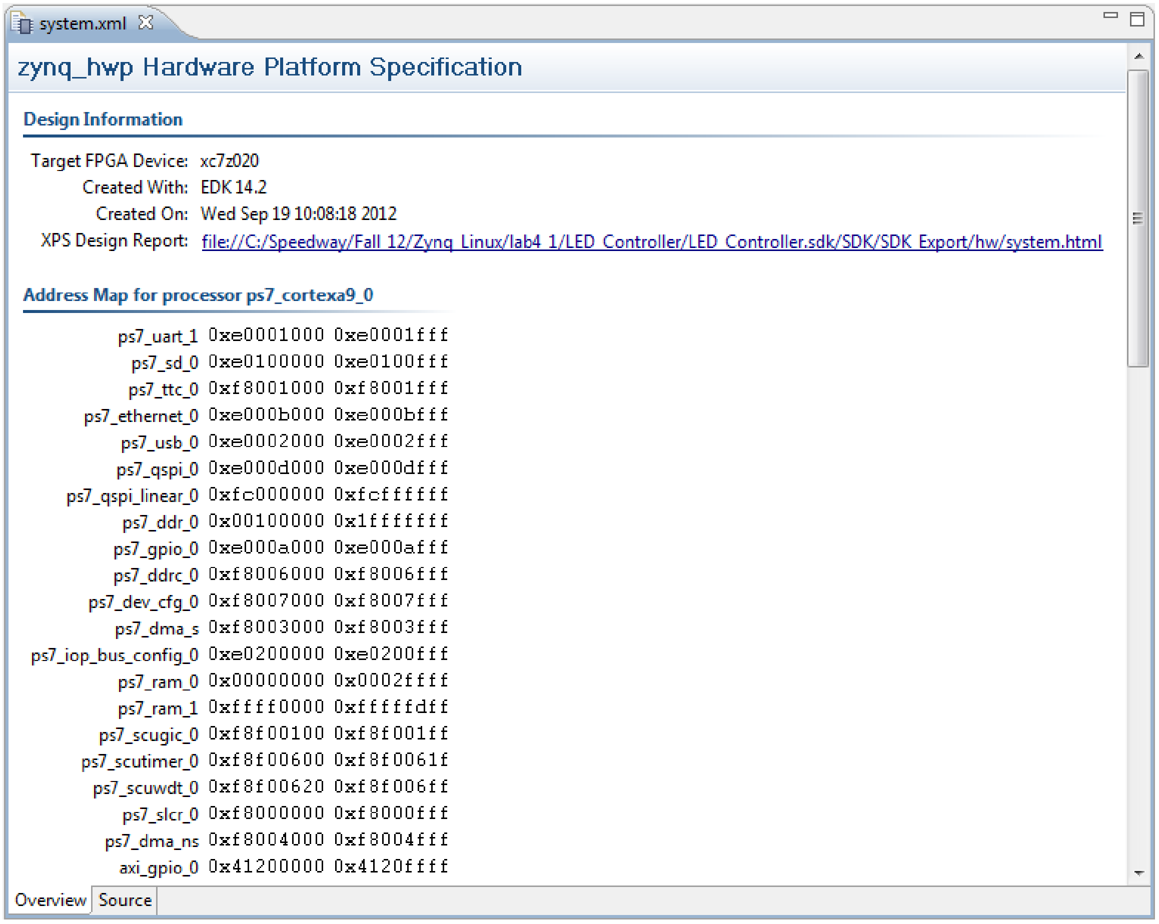
☐ Choose one of the existing pre-defined hardware platforms

Available pre-defined hardware platforms:

ZC702_hw_platform

Figure 5 – Entering Information to Create a New Hardware Project

6. The hardware platform specification (system.xml file) will be displayed. The system.xml is essentially an XML version of the XPS project (system.xmp) that was used in Lab 1.1 and the associated netlist (system.mhs) files.



zynq_hwp Hardware Platform Specification

Design Information

Target FPGA Device:	xc7z020
Created With:	EDK 14.2
Created On:	Wed Sep 19 10:08:18 2012
XPS Design Report:	file://C:/Speedway/Fall 12/Zynq Linux/lab4 1/LED Controller/LED Controller.sdk/SDK/SDK Export/hw/system.html

Address Map for processor ps7_cortexa9_0

ps7_uart_1	0xe0001000	0xe0001fff
ps7_sd_0	0xe0100000	0xe0100fff
ps7_ttc_0	0xf8001000	0xf8001fff
ps7_ethernet_0	0xe000b000	0xe000bfff
ps7_usb_0	0xe0002000	0xe0002fff
ps7_qspi_0	0xe000d000	0xe000dfff
ps7_qspi_linear_0	0xfc000000	0xfcffffff
ps7_ddr_0	0x00100000	0x1fffffff
ps7_gpio_0	0xe000a000	0xe000afff
ps7_ddrc_0	0xf8006000	0xf8006fff
ps7_dev_cfg_0	0xf8007000	0xf8007fff
ps7_dma_s	0xf8003000	0xf8003fff
ps7_iop_bus_config_0	0xe0200000	0xe0200fff
ps7_ram_0	0x00000000	0x0002ffff
ps7_ram_1	0xffff0000	0xfffffdff
ps7_scugic_0	0xf8f00100	0xf8f001ff
ps7_scutimer_0	0xf8f00600	0xf8f0061f
ps7_scuwdt_0	0xf8f00620	0xf8f006ff
ps7_slcr_0	0xf8000000	0xf8000fff
ps7_dma_ns	0xf8004000	0xf8004fff
axi_gpio_0	0x41200000	0x4120ffff

Figure 6 – Newly Created Hardware Project

While SDK supports multiple hardware platform projects in the same workspace, it is good practice to have one hardware platform per workspace.

Note: When browsing through the system.xml file, you can view information about the IP by clicking the adjacent link to the datasheet for the IP.

7. SDK manages software application projects within the workspace context so a new Linux application project must be added to the workspace. Once the new application project is added, we can begin adding source code that will allow us to access the led-brightness driver.

   Create a new SDK Linux software application project by selecting the **File→New→Xilinx C Project** menu item.

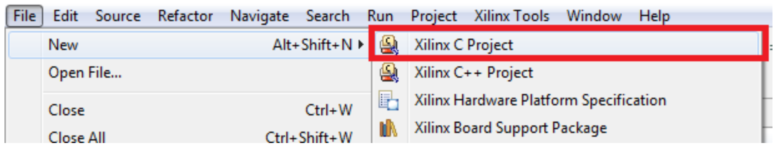
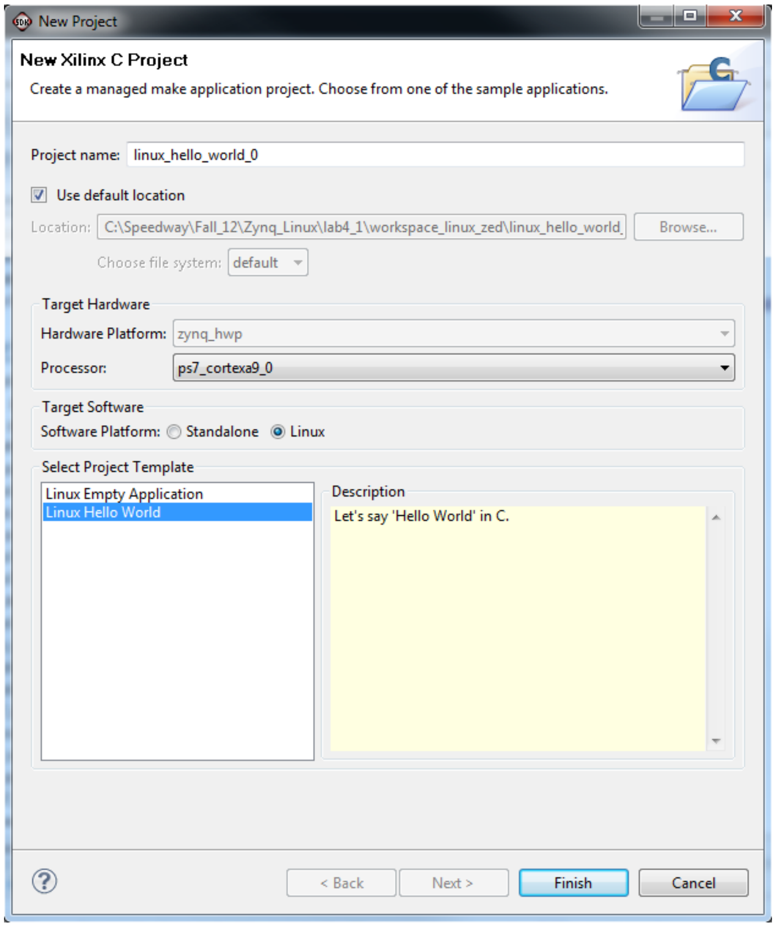


Figure 7 – Creating a C Application Project

8. Select **Linux** under the **Software Platform** options and select the **Linux Hello World** project template. Selecting the **Linux** Target Software option will update the default **Project name** entry to **linux_hello_world_0** which will be used for the new project.

   Setting the **Software Platform** option as **Linux** also enables the tool to run the Linux compiler.

   Click the **Finish** button to complete the new project creation process using the **Linux Hello World** project template.



New Project

New Xilinx C Project

Create a managed make application project. Choose from one of the sample applications.

Project name: linux_hello_world_0

Use default location [✓]

Location: C:\Speedway\Fall_12\Zynq_Linux\lab4_1\workspace_linux_zed\linux_hello_world_

Choose file system: default

Target Hardware

Hardware Platform: zynq_hwp

Processor: ps7_cortexa9_0

Target Software

Software Platform: Standalone [✓ Linux]

Select Project Template

Linux Empty Application

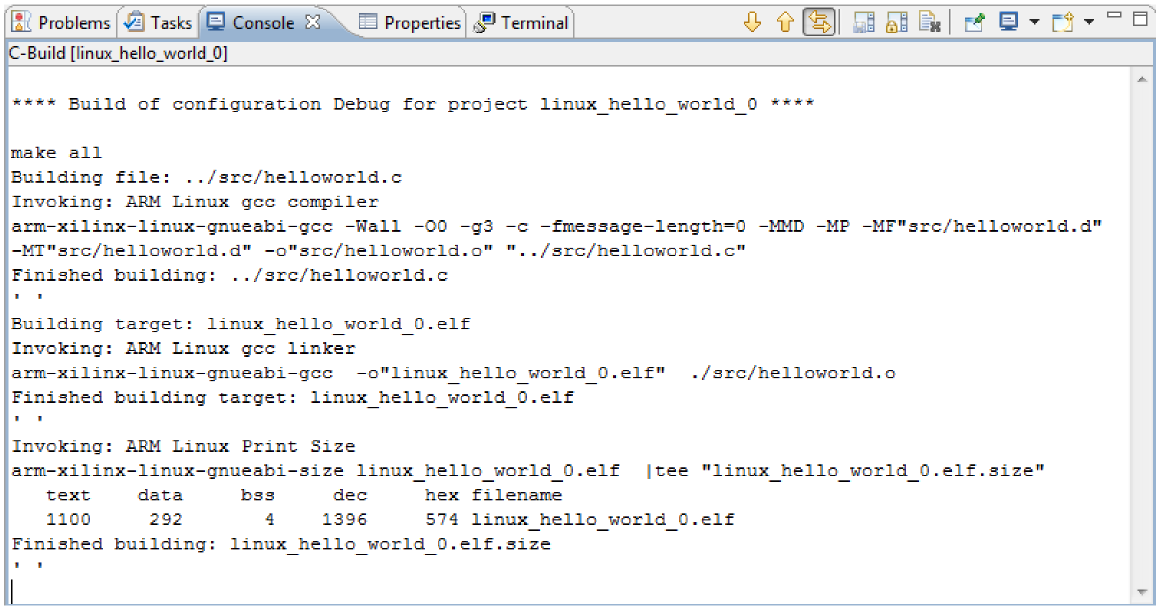
Linux Hello World

Description

Let's say 'Hello World' in C.

Figure 8 – Creating a Hello World Application

9. The application will be automatically built. The SDK **Console** panel shows the results of the build. Make sure that the application is built without errors.



```
**** Build of configuration Debug for project linux_hello_world_0 ****

make all
Building file: ../src/helloworld.c
Invoking: ARM Linux gcc compiler
arm-xilinx-linux-gnueabi-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MMD -MP -MF"src/helloworld.d"
-MT"src/helloworld.d" -o"src/helloworld.o" "../src/helloworld.c"
Finished building: ../src/helloworld.c
' '
Building target: linux_hello_world_0.elf
Invoking: ARM Linux gcc linker
arm-xilinx-linux-gnueabi-gcc  -o"linux_hello_world_0.elf"  ./src/helloworld.o
Finished building target: linux_hello_world_0.elf
' '
Invoking: ARM Linux Print Size
arm-xilinx-linux-gnueabi-size linux_hello_world_0.elf  |tee "linux_hello_world_0.elf.size"
   text    data     bss     dec     hex filename
   1100     292       4    1396     574 linux_hello_world_0.elf
Finished building: linux_hello_world_0.elf.size
' '
```

Figure 9 – Application Build Console Window

10. Connect 12 V power supply to ZedBoard barrel jack (J20).

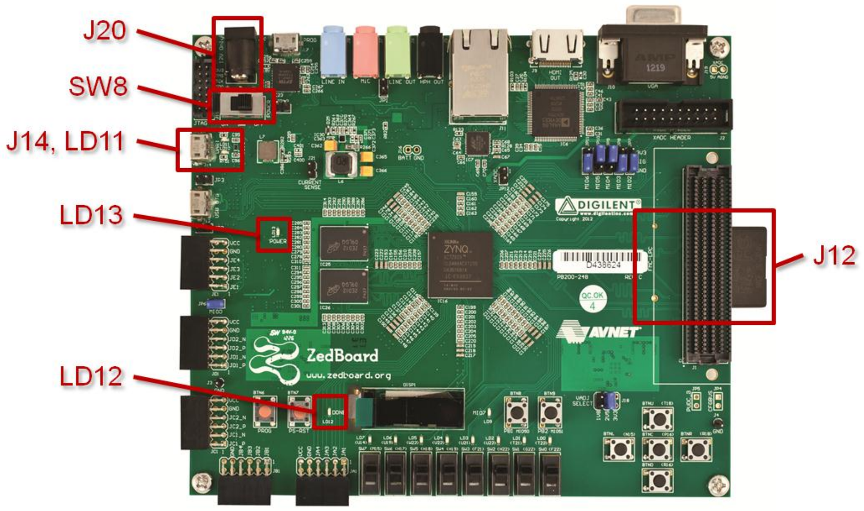


Figure 10 – ZedBoard Hardware Reference

11. Connect the USB-UART port of ZedBoard (J14) which is labeled UART to a PC using the MicroUSB cable.

12. Insert the 4GB SD card included with ZedBoard into the SD card slot (J12) located on the underside of ZedBoard PCB.

13. Connect an Ethernet patch cable between the development PC network adapter and ZedBoard Ethernet jack J11.

    Note that the default static IP address of ZedBoard is 192.168.1.10 so the IP address of the network adapter on the development PC should be set to another non-conflicting IP address in the range 192.168.1.1 to 192.168.1.254.

14. Verify the ZedBoard boot mode (JP7-JP11) and MIO0 (JP6) jumpers are set to SD card mode as described in the Hardware Users Guide:

    http://www.zedboard.org/sites/default/files/ZedBoard_HW_UG_v1_6.pdf

    A copy of the Hardware Users Guide is also located in the SpeedWay **C:\Speedway\Fall_12\Zynq_Linux\support_documents\** folder for your convenience.

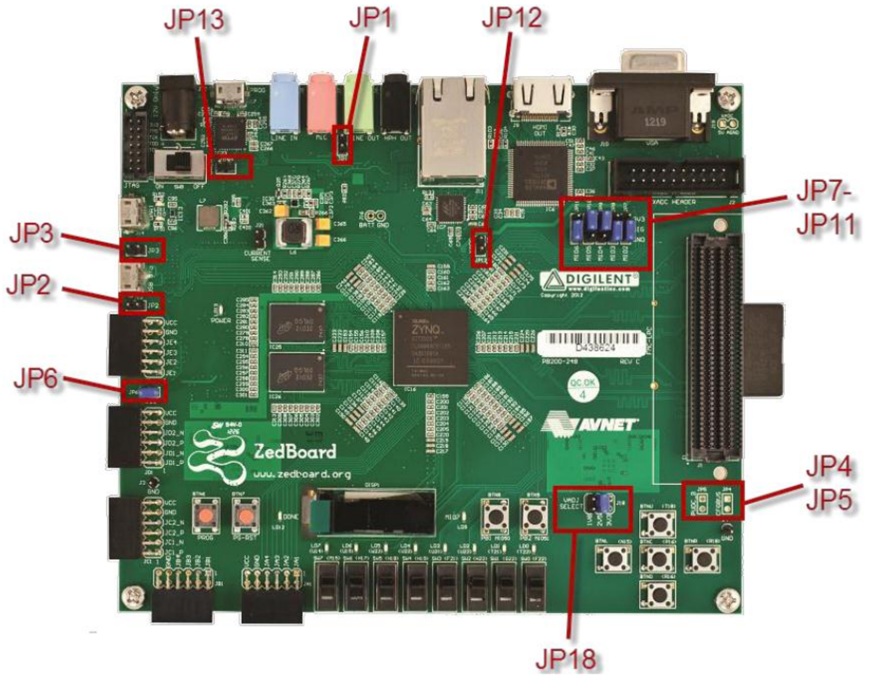


Figure 11 – ZedBoard Jumper Settings

15. Turn power switch (SW8) to the ON position. ZedBoard will power on and the Green Power Good LED (LD13) should illuminate.

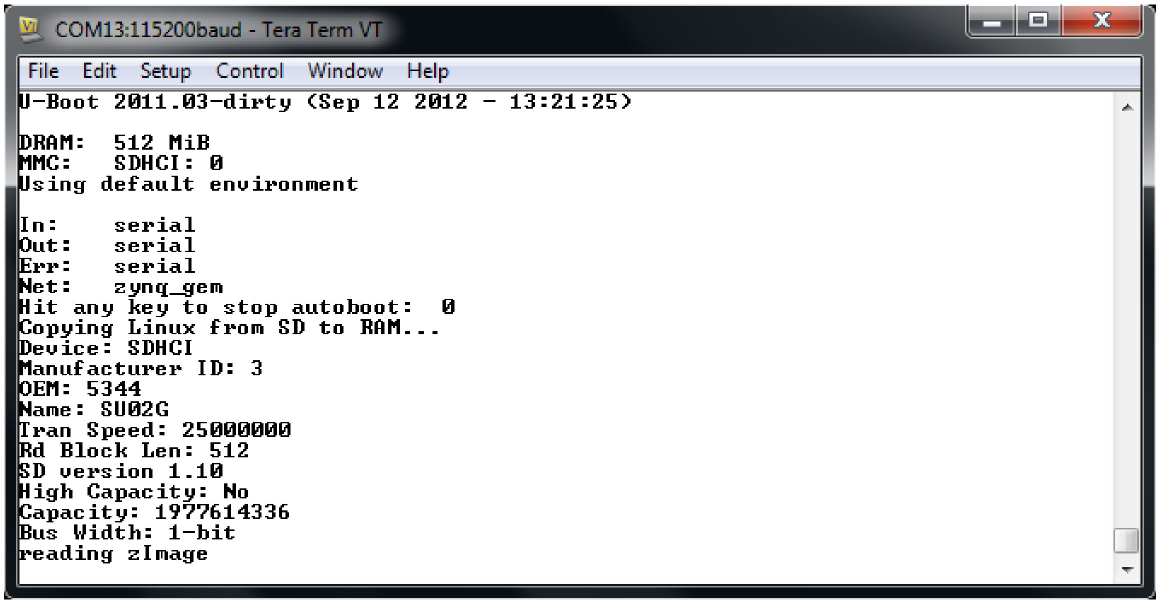
16. Wait approximately 15 seconds. The blue Done LED (LD12) should illuminate.

17. On the PC, if a serial terminal session is not already open, open a serial terminal program. Tera Term was used to show the example output for this lab document.

Figure 12 – Tera Term Icon

18. If the amber USB-Link Status (LD11) does not flicker during boot to indicate activity, check the driver installation to determine if the device driver is recognized and enumerated successfully and that there are no errors reported by Windows.

19. Power cycle the ZedBoard and monitor the Tera Term window. When the terminal output from U-Boot and a countdown is observed, allow the countdown to expire.


COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
U-Boot 2011.03-dirty (Sep 12 2012 - 13:21:25)

DRAM:  512 MiB
MMC:   SDHCI: 0
Using default environment

In:    serial
Out:   serial
Err:   serial
Net:   zynq_gem
Hit any key to stop autoboot:  0
Copying Linux from SD to RAM...
Device: SDHCI
Manufacturer ID: 3
OEM: 5344
Name: SU02G
Tran Speed: 25000000
Rd Block Len: 512
SD version 1.10
High Capacity: No
Capacity: 1977614336
Bus Width: 1-bit
reading zImage
```

Figure 13 – ZedBoard U-Boot Booting Linux

20. When the Linux command prompt is reached, return to SDK and select the **Window→Open Perspective→Other** menu item to open a new SDK perspective.

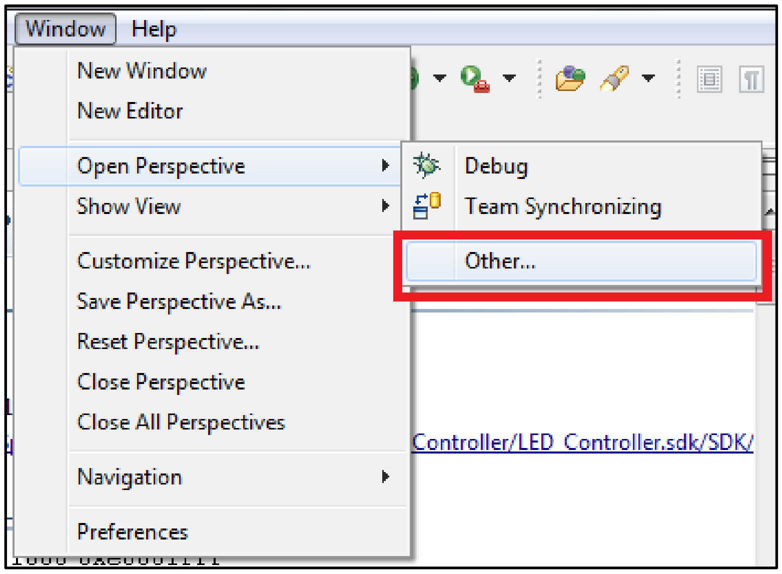


Figure 14 – Selecting Open Perspective - Other

21. Select the **Remote System Explorer** option and then click the **OK** button.

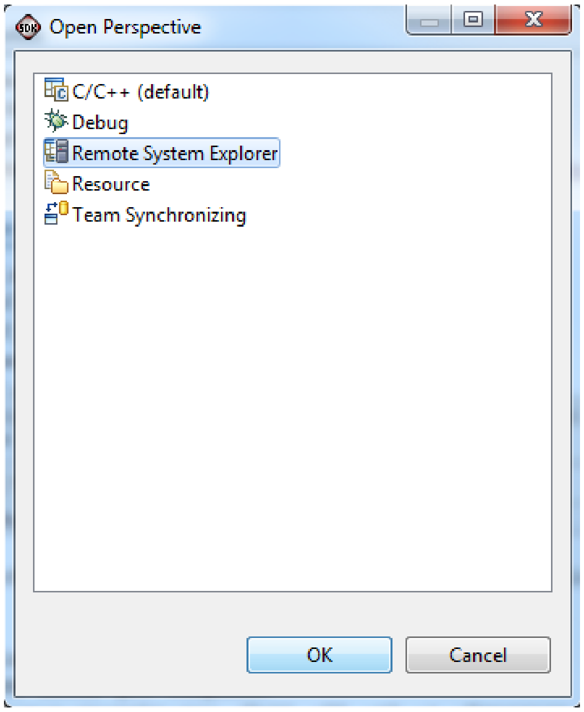


Figure 15 – Selecting Remote System Explorer

22. After the new RSE perspective opens, establish a new connection by right-clicking on the **Local** node in the **Remote Systems** panel and selecting the **New→Connection** menu item.

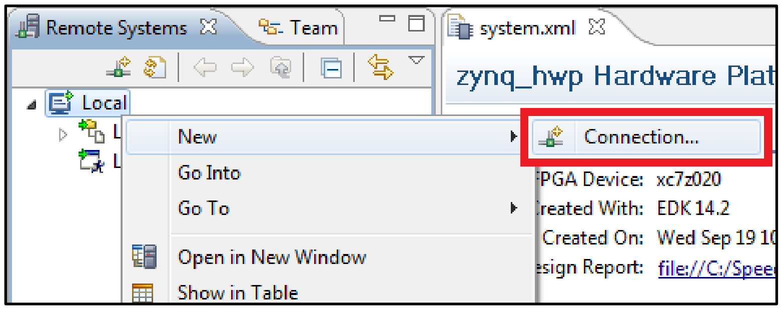


Figure 16 – Selecting New Connection

23. In the Select Remote System Type dialog box, select the **SSH Only** option and then click the **Next** button.

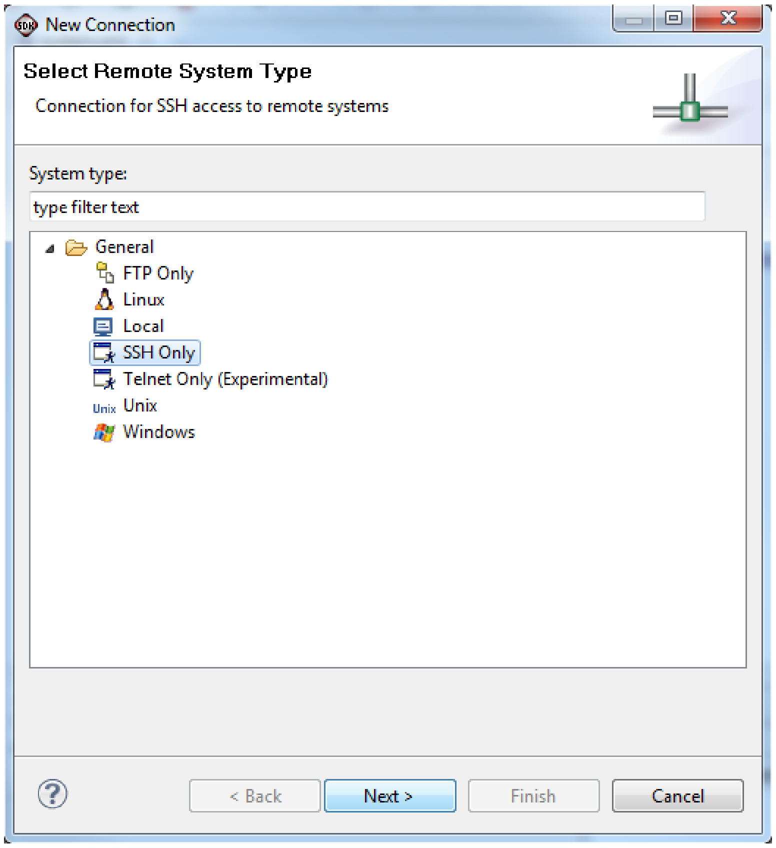
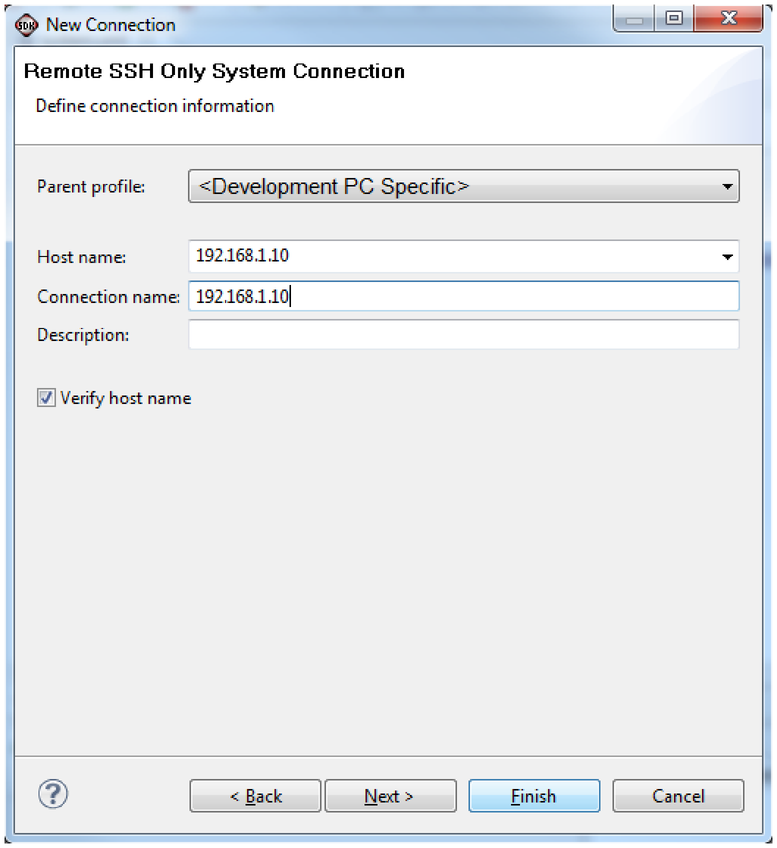


Figure 17 – Creating a New SSH RSE Connection

24. Select the IP address for the new connection by entering **192.168.1.10** in both the **Host name** and **Connection name** fields. The 192.168.1.10 IP address is the static IP set by the Linux kernel (specified in the device tree binary file on the SD card) for ZedBoard.

    Leave the **Parent profile** field at its default setting (which will be specific to your development machine configuration) and click the **Finish** button.


New Connection

Remote SSH Only System Connection

Define connection information

Parent profile: <Development PC Specific>

Host name: 192.168.1.10

Connection name: 192.168.1.10

Description:

☑ Verify host name

Figure 18 – Entering the Host and Connection Names

25. The new connection is now present in the **Remote Systems** tab and the target Linux root file system can be accessed via this connection. Browse the directory structure of the **192.168.1.10** connection by expanding the **192.168.1.10** entry and the **Sftp Files** subfolder in the remote systems panel.

    Expand the **Root** entry and after a few seconds, the tree will populate with a file system structure as seen in Figure 19.

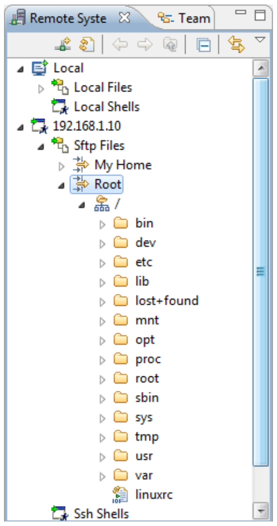
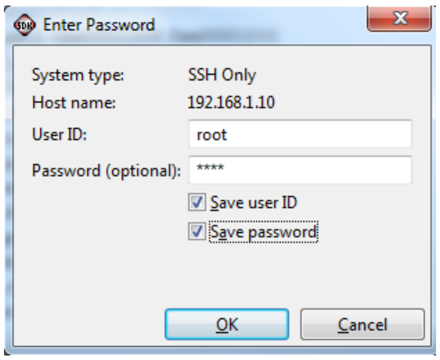


Figure 19 – Entering the Login Information for the Remote Connection

This is the Linux file system in the DDR3 RAM of the ZC702 or ZED board. The **Local** connection is the directory structure of the host development machine. You can even drag-and-drop files between the two connections or other windows directory panes.

26. If prompted for a **User ID** and **Password**, use the entry **root** for both of these fields. Also, for future convenience, select the Save user ID and Save password options.

    If prompted to establish the authenticity of the remote host 192.168.1.10, click on the **Yes** button to continue initiating the connection.

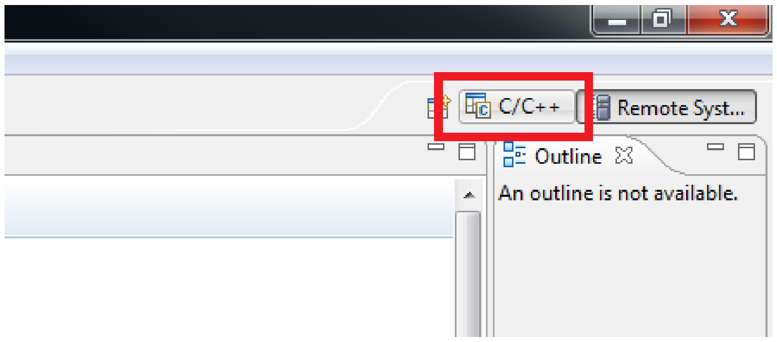


Enter Password

System type: SSH Only
Host name: 192.168.1.10
User ID: root
Password (optional): ****
[✓] Save user ID
[✓] Save password

Figure 20 – Entering the Login Information for the Remote Connection

27. Click the **C/C++** perspective tab button (top right corner of SDK window) to return to the original perspective. If the perspective tab buttons are hidden, click on the **>>** button to reveal a dropdown selection and then select the **C/C++** option from the list.

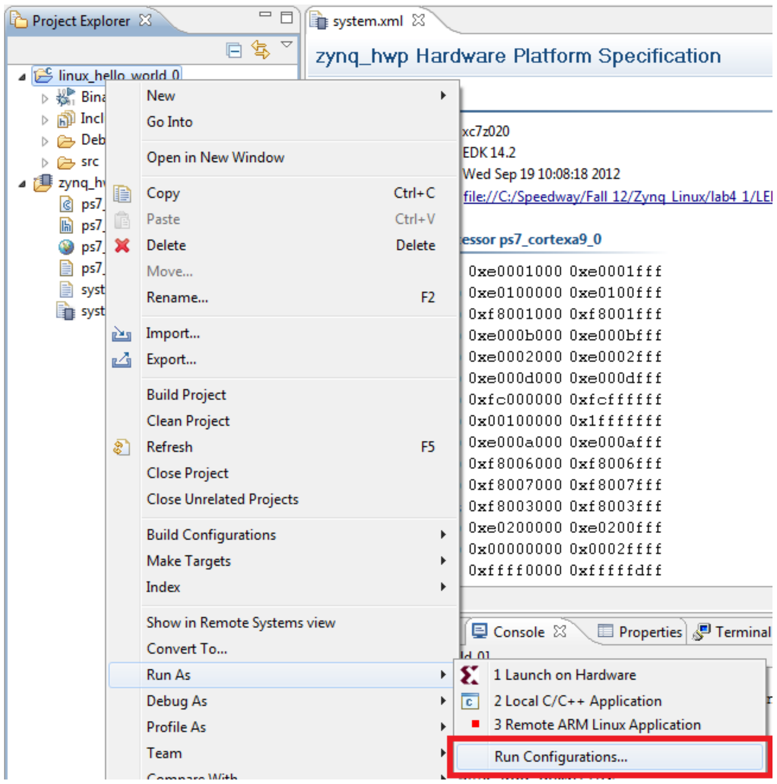


An outline is not available.

Figure 21 – Returning to the C/C++ Perspective

28. A new application run configuration must be created. Run configurations associate an ELF object file to a target for execution. In this case, the target is a hardware board accessed over a network TCP/IP connection.

29. In the **Project Explorer** tab, right-click the **linux_hello_world_0** project and select the **Run As→Run Configurations...** menu item.



```
xc7z020
EDK 14.2
Wed Sep 19 10:08:18 2012
file://C:/Speedway/Fall 12/Zynq Linux/lab4 1/LE
essor ps7_cortexa9_0
0xe0001000 0xe0001fff
0xe0100000 0xe0100fff
0xf8001000 0xf8001fff
0xe000b000 0xe000bfff
0xe0002000 0xe0002fff
0xe000d000 0xe000dfff
0xfc000000 0xfcffffff
0x00100000 0x1fffffff
0xe000a000 0xe000afff
0xf8006000 0xf8006fff
0xf8007000 0xf8007fff
0xf8003000 0xf8003fff
0xe0200000 0xe0200fff
0x00000000 0x0002ffff
0xffff0000 0xfffffdff
```

Figure 22 – Selecting the Application Run Configurations

30. Keep in mind that that this application is a Linux application rather than a Standalone/bare-metal application. As a result, the Run Configuration process is different than that used in the Zynq Intro SpeedWay exercises.

    From the Create, manage, and run configurations dialog box, double-click the **Remote ARM Linux Application** option.

    Select **192.168.1.10** from the **Connection** drop-down list.

    The default **Project** name is pre-populated as **linux_hello_world_0**, leave this entry at the default setting.

    The default **Build Configuration** is automatically set to **Debug** which is exactly what we want for our very first application launch.

    The application must reside in the target root file system in order for it to be executed. Select a remote temporary location to store the application in by setting the **Remote Absolute File Path for C/C++ Application** field by clicking the adjacent **Browse** button, then select **192.168.1.10** as the **Connection**, expand the **Root** folder to locate the **/tmp** subfolder, and then click the **OK** button.

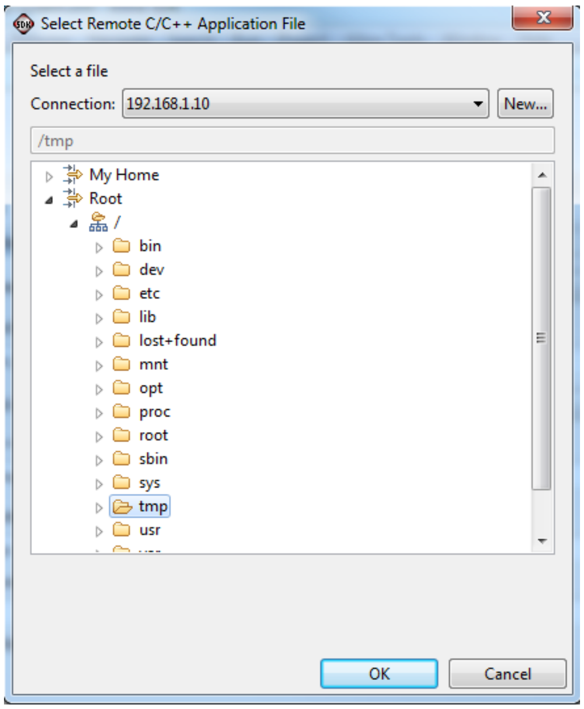


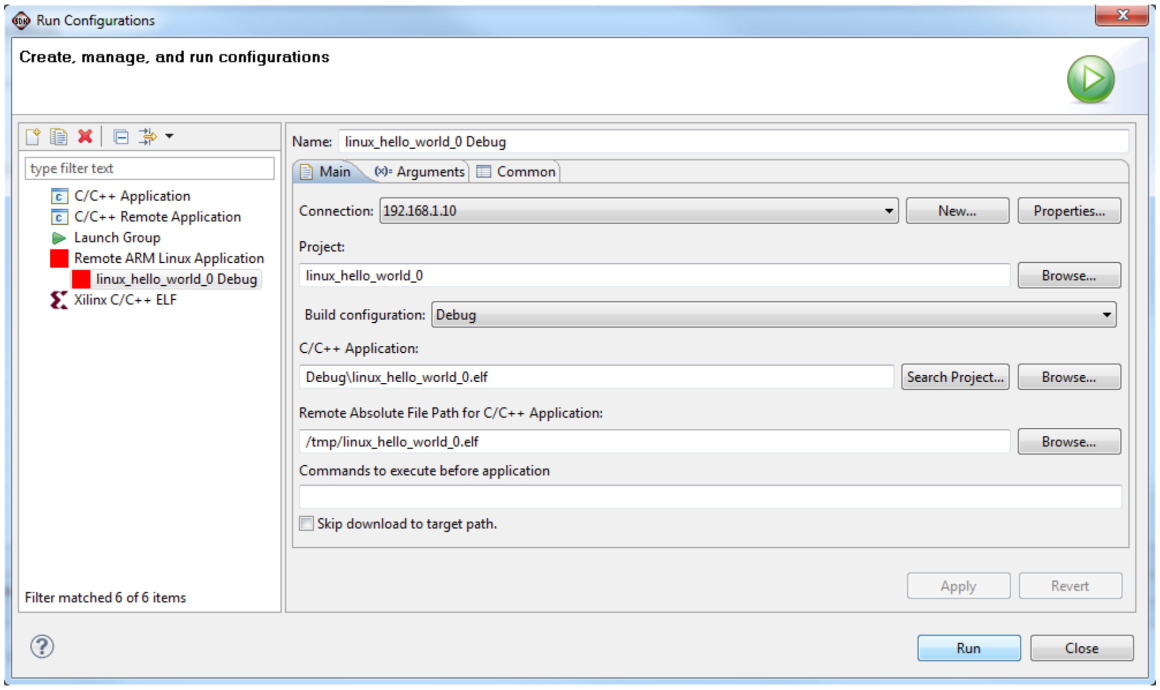
Figure 23 – Setting the Linux Application Run Configuration

If prompted for user credentials, use the same options from Step 26.

In the **Remote Absolute File Path for C/C++ Application** field, modify the entry to **/tmp/linux_hello_world_0.elf** as the target file path. This will place and rename the **linux_hello_world_0.elf** file into the **/tmp** directory of the target.

Once the settings are in place, click the **Apply** button to force the changes into effect.

Once the changes are applied then launch the run configuration by clicking on the **Run** button.



Name: linux_hello_world_0 Debug

Connection: 192.168.1.10

Project: linux_hello_world_0

Build configuration: Debug

C/C++ Application: Debug\linux_hello_world_0.elf

Remote Absolute File Path for C/C++ Application: /tmp/linux_hello_world_0.elf

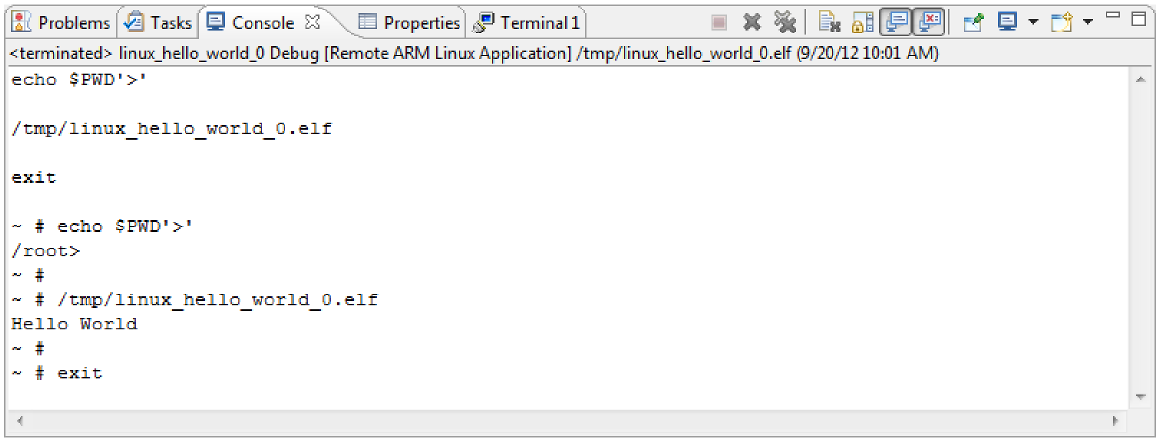
Commands to execute before application

☐ Skip download to target path.

Figure 24 – Setting the Linux Application Run Configuration

31. After the application run configuration launches, observe that the application output is displayed in the SDK Console window.

    Note that this SDK Console output is captured across the SSH terminal connection to the target board rather than the serial connection that has been used in previous labs.

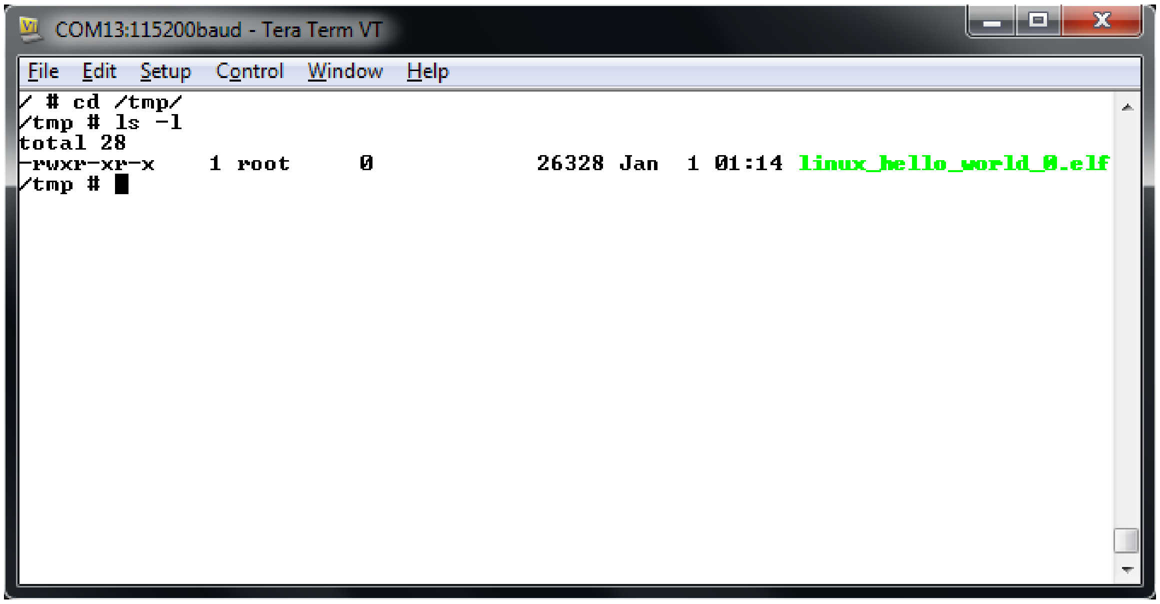


```
<terminated> linux_hello_world_0 Debug [Remote ARM Linux Application] /tmp/linux_hello_world_0.elf (9/20/12 10:01 AM)
echo $PWD'>'

/tmp/linux_hello_world_0.elf

exit

~ # echo $PWD'>'
/root>
~ #
~ # /tmp/linux_hello_world_0.elf
Hello World
~ #
~ # exit
```

Figure 25 – Application Output in SDK Console

32. Switch back to the ZedBoard serial console in TeraTerm. Change the working directory to the **/tmp** folder and then perform a directory listing to verify that the application executable is indeed in the **/tmp** folder and it has executable permissions.

```
# cd /tmp/

# ls *
```


COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help

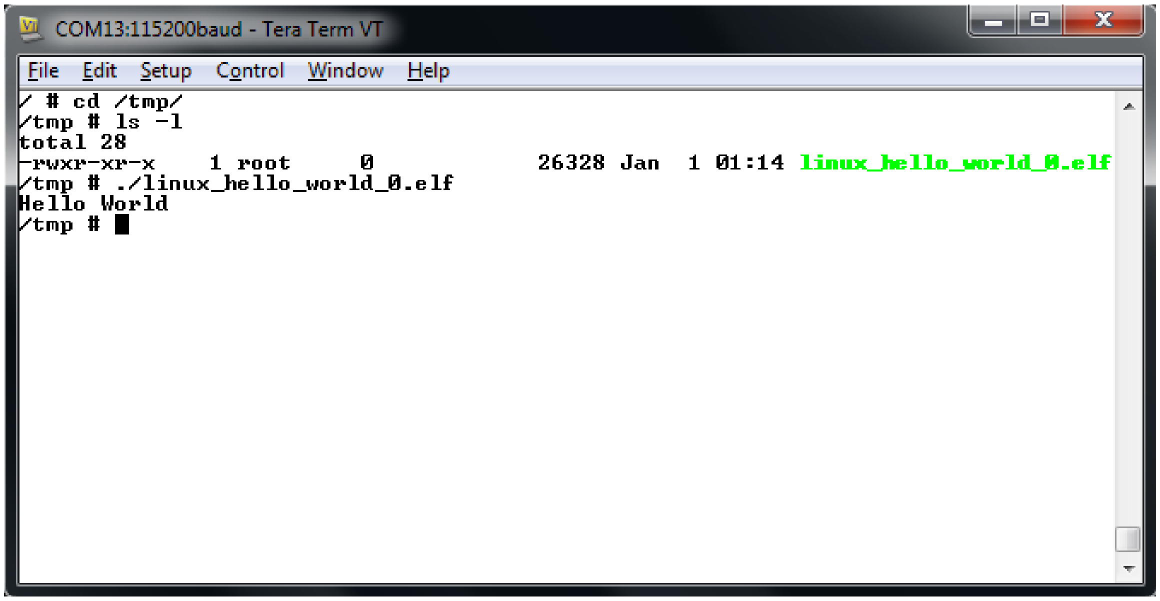
```
/ # cd /tmp/
/tmp # ls -l
total 28
-rwxr-xr-x    1 root     0           26328 Jan  1 01:14 linux_hello_world_0.elf
/tmp # 
```

Figure 26 – Hello World Application is Located in the /tmp Folder

33. Execute the Hello World application and observe the output.

```
# ./linux_hello_world_0.elf
```

Notice that the output of the `printf()` system call is shown on the terminal.


COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
/ # cd /tmp/
/tmp # ls -l
total 28
-rwxr-xr-x    1 root     0           26328 Jan  1 01:14 linux_hello_world_0.elf
/tmp # ./linux_hello_world_0.elf
Hello World
/tmp #
```

Figure 27 – Application Output in Serial Terminal

34. Click the **C/C++** perspective tab (top right corner of SDK window) to return to the original perspective.

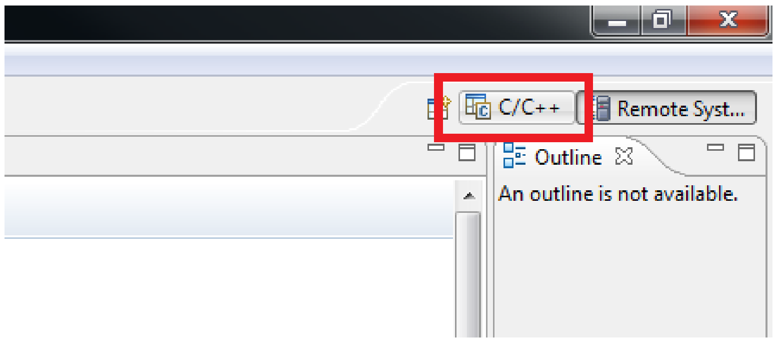


Figure 28 – Returning to the C/C++ Perspective

Questions:

Answer the following questions:

- *What communications mechanism is used to command the target ZedBoard from SDK?*

____

Experiment 2: Application Interaction with the Hardware

An application will be created which interacts with the ZedBoard hardware.

Experiment 2 General Instruction:

Boot ZedBoard using the SD card with the created Zynq boot image and observe the terminal output.

Experiment 2 Step-by-Step Instructions:

1. SDK manages software application projects within the workspace context so another new application project must be added to the workspace.

   Create a new SDK Linux software application project by selecting the **File→New→Xilinx C Project** menu item.

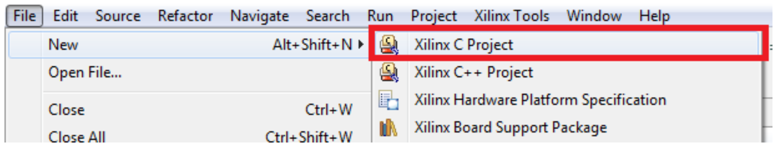
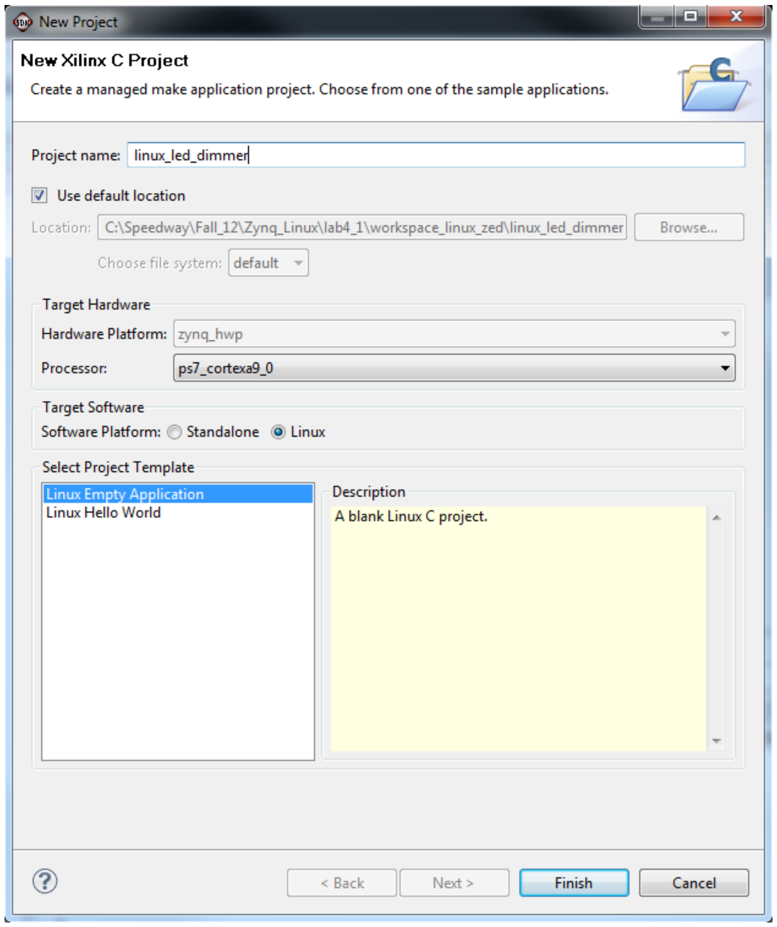


Figure 29 – Creating a C Application Project

2. Select **Linux** under the **Software Platform** options and select the **Linux Empty Application** project template. Selecting the **Linux** Target Software option will update the default **Project name** entry to **linux_empty_application_0** which we will change to **linux_led_dimmer** for the new project.

   Setting the **Software Platform** option as **Linux** also enables the tool to run the Linux compiler.

   Click the **Finish** button to complete the new project creation process using the **Linux Empty Application** project template.



New Xilinx C Project

Create a managed make application project. Choose from one of the sample applications.

Project name: linux_led_dimmer

☑ Use default location

Location: C:\Speedway\Fall_12\Zynq_Linux\lab4_1\workspace_linux_zed\linux_led_dimmer

Choose file system: default

Target Hardware

Hardware Platform: zynq_hwp

Processor: ps7_cortexa9_0

Target Software

Software Platform: ○ Standalone ● Linux

Select Project Template

Linux Empty Application
Linux Hello World

Description

A blank Linux C project.

Figure 30 – Creating a New Empty Application

3. The application will automatically try to build, but there are no source files in the project to be built yet.

   Begin adding necessary source files to the project. One way to do this is to follow the **File→Import** tool as was done in the Introduction to Zynq SpeedWay training lab exercises.

   This source code import can also be done with a time saving shortcut by locating the **linux_led_dimmer.c** file in the Lab 4.1 folder on the host operating system by using Windows Explorer to navigate to the following folder:

   C:\Speedway\Fall_12\Zynq_Linux\lab4_1\led_dimmer_src\

   Make sure the Windows Explorer window does not take up the entire screen and that the SDK Project Explorer panel is visible in the background. Select the source code file **linux_led_dimmer.c** and drag the file into the SDK **linux_led_dimmer** project **src** folder.

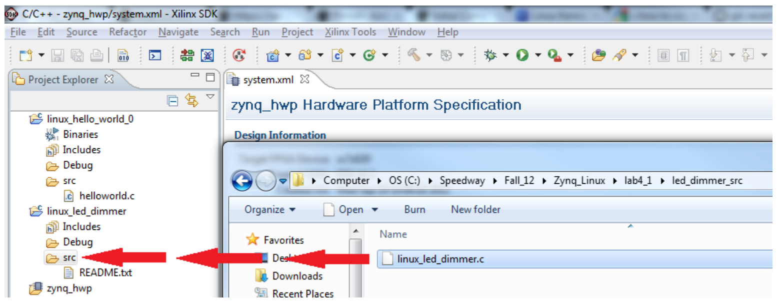
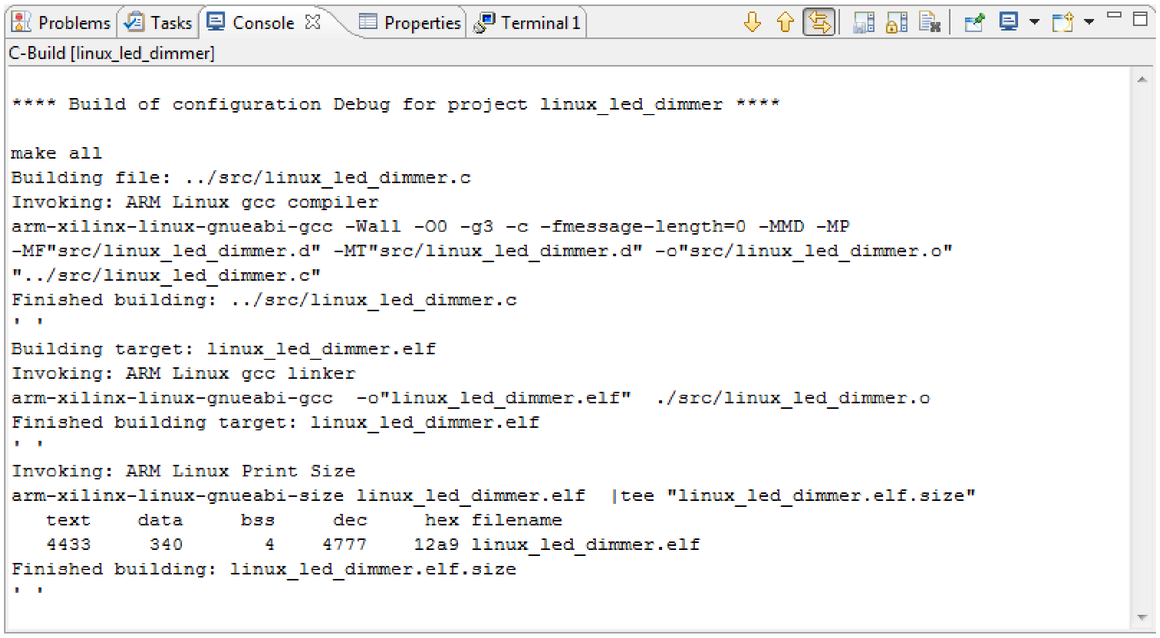

C/C++ - zynq_hwp/system.xml - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
Project Explorer
linux_hello_world_0
Binaries
Includes
Debug
src
helloworld.c
linux_led_dimmer
Includes
Debug
src
README.txt
zynq_hwp
system.xml
zynq_hwp Hardware Platform Specification
Design Information
Computer ▸ OS (C:) ▸ Speedway ▸ Fall_12 ▸ Zynq_Linux ▸ lab4_1 ▸ led_dimmer_src
Organize Open Burn New folder
Favorites
Desktop
Downloads
Recent Places
Name
linux_led_dimmer.c

Figure 31 – Importing Source Code Into SDK Application Project

4. The SDK **Console** panel shows the results of the build. Make sure that the application is built without errors.

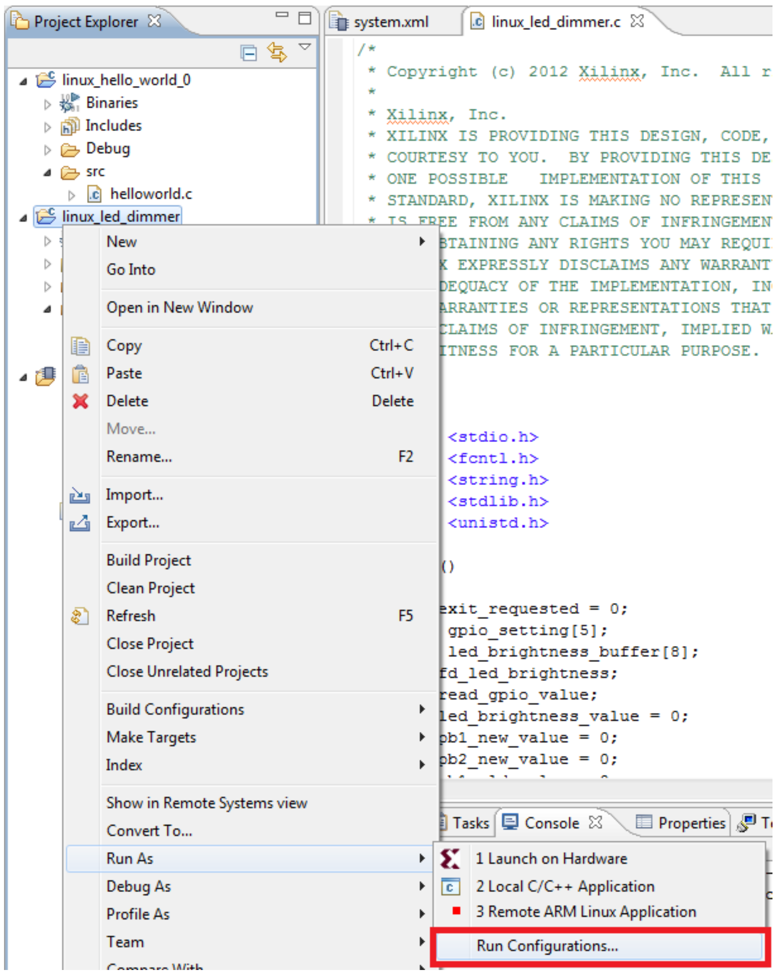


```
**** Build of configuration Debug for project linux_led_dimmer ****

make all
Building file: ../src/linux_led_dimmer.c
Invoking: ARM Linux gcc compiler
arm-xilinx-linux-gnueabi-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MMD -MP
-MF"src/linux_led_dimmer.d" -MT"src/linux_led_dimmer.d" -o"src/linux_led_dimmer.o"
"../src/linux_led_dimmer.c"
Finished building: ../src/linux_led_dimmer.c
' '
Building target: linux_led_dimmer.elf
Invoking: ARM Linux gcc linker
arm-xilinx-linux-gnueabi-gcc  -o"linux_led_dimmer.elf"  ./src/linux_led_dimmer.o
Finished building target: linux_led_dimmer.elf
' '
Invoking: ARM Linux Print Size
arm-xilinx-linux-gnueabi-size linux_led_dimmer.elf  |tee "linux_led_dimmer.elf.size"
   text    data     bss     dec     hex filename
   4433     340       4    4777    12a9 linux_led_dimmer.elf
Finished building: linux_led_dimmer.elf.size
' '
```

Figure 32 – Application Build Console Window

5. A new application run configuration must be created. Run configurations associate an ELF object file to a target for execution. In this case, the target is a hardware board accessed over a network TCP/IP connection.

   In the **Project Explorer** tab, right-click the **linux_led_dimmer** project and select the **Run As→Run Configurations...** menu item.



```
/*
 * Copyright (c) 2012 Xilinx, Inc.  All r
 *
 * Xilinx, Inc.
 * XILINX IS PROVIDING THIS DESIGN, CODE,
 * COURTESY TO YOU.  BY PROVIDING THIS DE
 * ONE POSSIBLE   IMPLEMENTATION OF THIS
 * STANDARD, XILINX IS MAKING NO REPRESEN
 * IS FREE FROM ANY CLAIMS OF INFRINGEMEN
BTAINING ANY RIGHTS YOU MAY REQUI
X EXPRESSLY DISCLAIMS ANY WARRANT
DEQUACY OF THE IMPLEMENTATION, IN
ARRANTIES OR REPRESENTATIONS THAT
CLAIMS OF INFRINGEMENT, IMPLIED W
ITNESS FOR A PARTICULAR PURPOSE.

 <stdio.h>
 <fcntl.h>
 <string.h>
 <stdlib.h>
 <unistd.h>

()

exit_requested = 0;
 gpio_setting[5];
 led_brightness_buffer[8];
fd_led_brightness;
read_gpio_value;
led_brightness_value = 0;
pb1_new_value = 0;
pb2_new_value = 0;
```

Figure 33 – Selecting the Application Run Configurations

6. From the Create, manage, and run configurations dialog box, double-click the **Remote ARM Linux Application** option

   Select **192.168.1.10** from the **Connection** drop-down list.

   The default **Project** name is pre-populated as **linux_led_dimmer**, leave this entry at the default setting.

   The default **Build Configuration** is automatically set to **Debug** which is exactly what we want for our application launch.

   The application must reside in the target root file system in order for it to be executed. Select a remote temporary location to store the application in by setting the **Remote Absolute File Path for C/C++ Application** field by clicking the adjacent **Browse** button, then select **192.168.1.10** as the **Connection**, expand the **Root** folder to locate the **/tmp** subfolder, and then click the **OK** button.

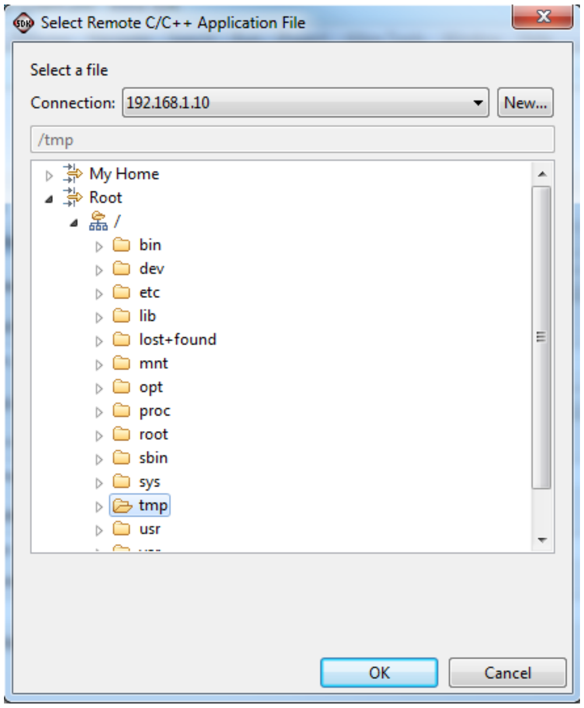


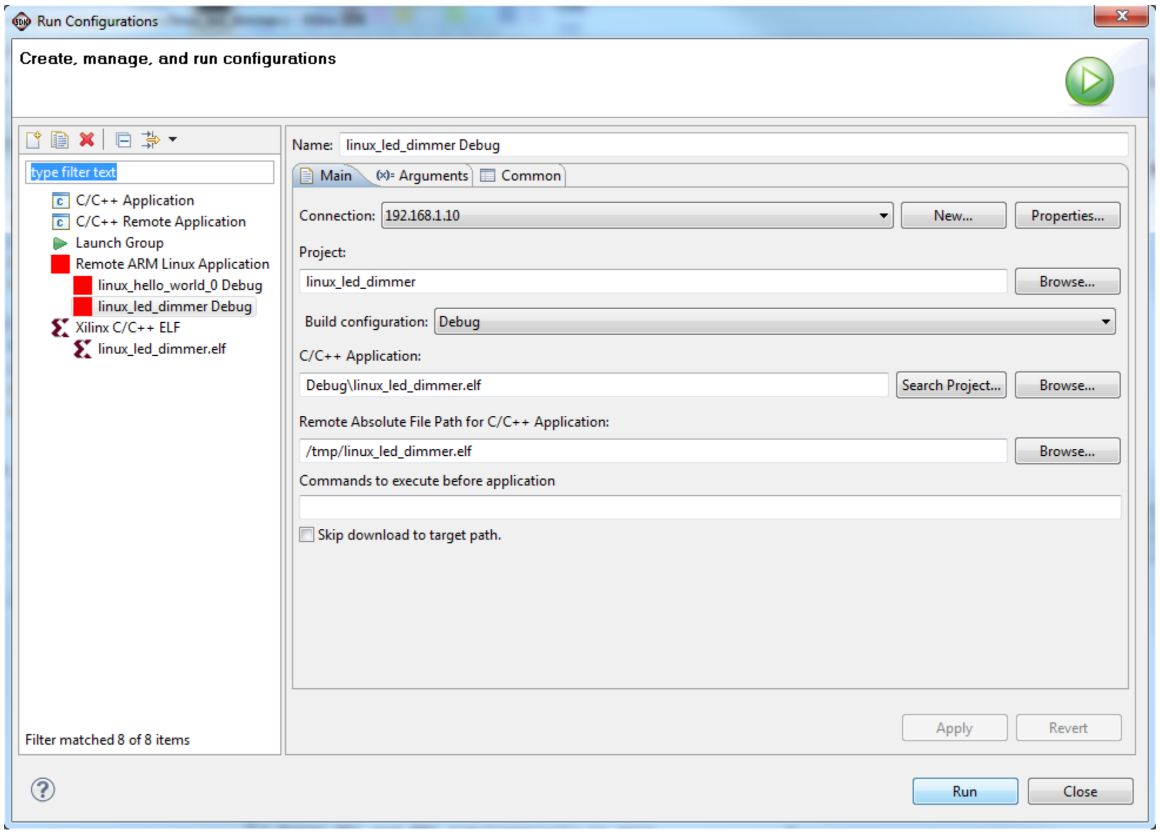
Figure 34 – Setting the Linux Application Run Configuration

   If prompted for user credentials, use the same options from Exercise1, Step 26.

   In the **Remote Absolute File Path for C/C++ Application** field, enter **/tmp/linux_led_dimmer.elf** as the target file path. This will place and rename the **linux_led_dimmer.elf** file into the **/tmp** directory of the target.

Once the settings are in place, click the **Apply** button to force the changes into effect.

Once the changes are applied then launch the run configuration by clicking on the **Run** button.


Run Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- Launch Group
- Remote ARM Linux Application
  - linux_hello_world_0 Debug
  - linux_led_dimmer Debug
- Xilinx C/C++ ELF
  - linux_led_dimmer.elf

Filter matched 8 of 8 items

Name: linux_led_dimmer Debug

Main | Arguments | Common

Connection: 192.168.1.10

Project: linux_led_dimmer

Build configuration: Debug

C/C++ Application: Debug\linux_led_dimmer.elf

Remote Absolute File Path for C/C++ Application: /tmp/linux_led_dimmer.elf

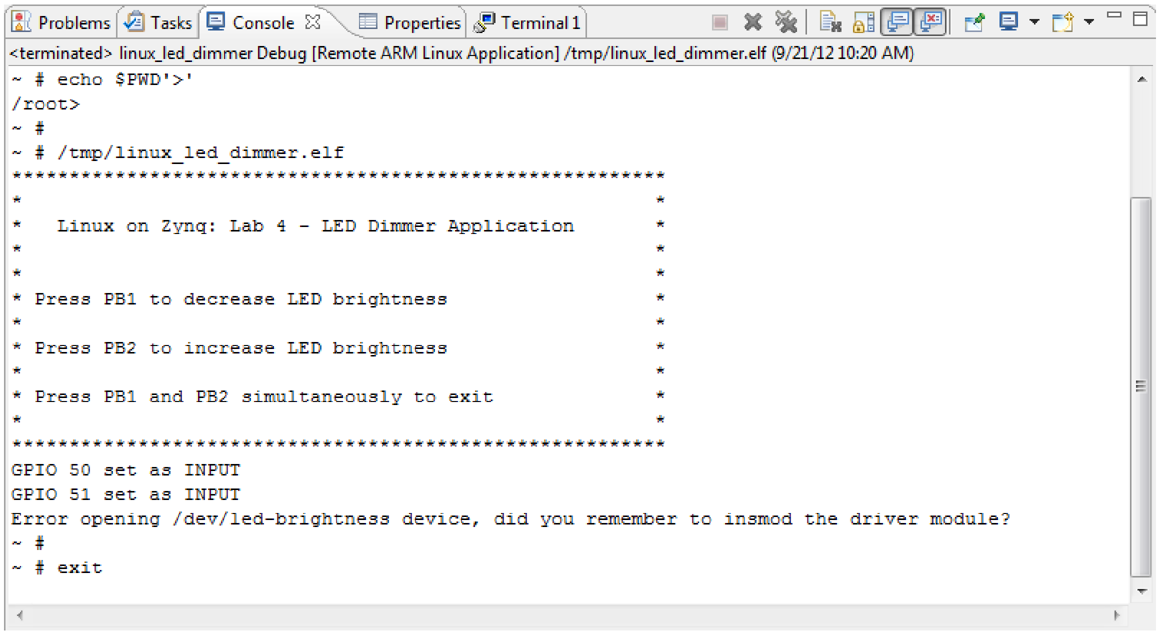
Commands to execute before application

Skip download to target path.

Figure 35 – Setting the Linux Application Run Configuration

7. After the application run configuration launches, observe that the application output is displayed in the SDK Console window.

   Notice that an error message may be shown in the application console output that the **/dev/led-brightness** device was not inserted. If this error message is shown, proceed to the next steps to insert the driver module manually. If the error message is not shown, skip ahead to Step 12.



```
<terminated> linux_led_dimmer Debug [Remote ARM Linux Application] /tmp/linux_led_dimmer.elf (9/21/12 10:20 AM)
~ # echo $PWD'>'
/root>
~ #
~ # /tmp/linux_led_dimmer.elf
*********************************************************
*                                                       *
*   Linux on Zynq: Lab 4 - LED Dimmer Application       *
*                                                       *
*                                                       *
* Press PB1 to decrease LED brightness                  *
*                                                       *
* Press PB2 to increase LED brightness                  *
*                                                       *
* Press PB1 and PB2 simultaneously to exit              *
*                                                       *
*********************************************************
GPIO 50 set as INPUT
GPIO 51 set as INPUT
Error opening /dev/led-brightness device, did you remember to insmod the driver module?
~ #
~ # exit
```

Figure 36 – Application Output in SDK Console

8. To insert the led-brightness driver module into the kernel, first mount the FAT32 file system on the SD card.

   The first partition on the SD card shows up as mmcblk0p1 in the device tree. This device represents MMC block device 0, partition 1. We will mount this device to the mount point /mnt and change the working directory to that folder so that the FAT32 file system can be accessed.

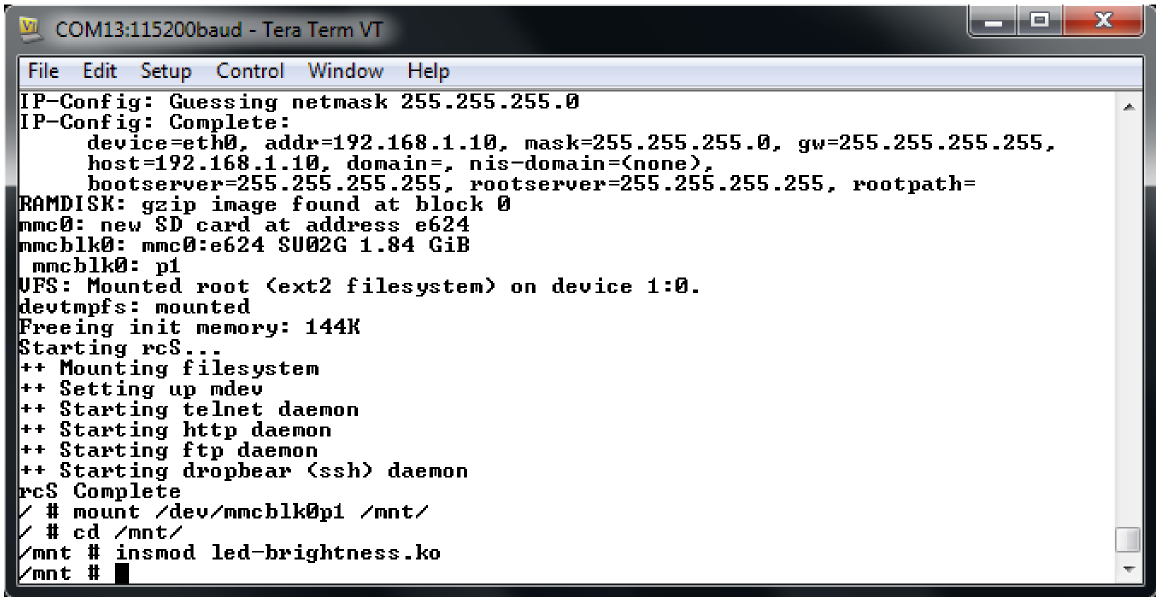
   From the serial terminal (Tera Term) window, enter the following commands:

```
# mount /dev/mmcblk0p1 /mnt/

# cd /mnt/
```

9. Now that the FAT32 partition on the SD card has been mounted, the led-brightness driver module that was placed in Experiment 1 can be loaded into the kernel using the insmod command.

```
# insmod led-brightness.ko
```

The good news is that there were no error messages thrown by the kernel when inserting the driver module. However, notice that none of the debug statements our driver sends to the console (via the printk() kernel call) are shown on the terminal.


COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help

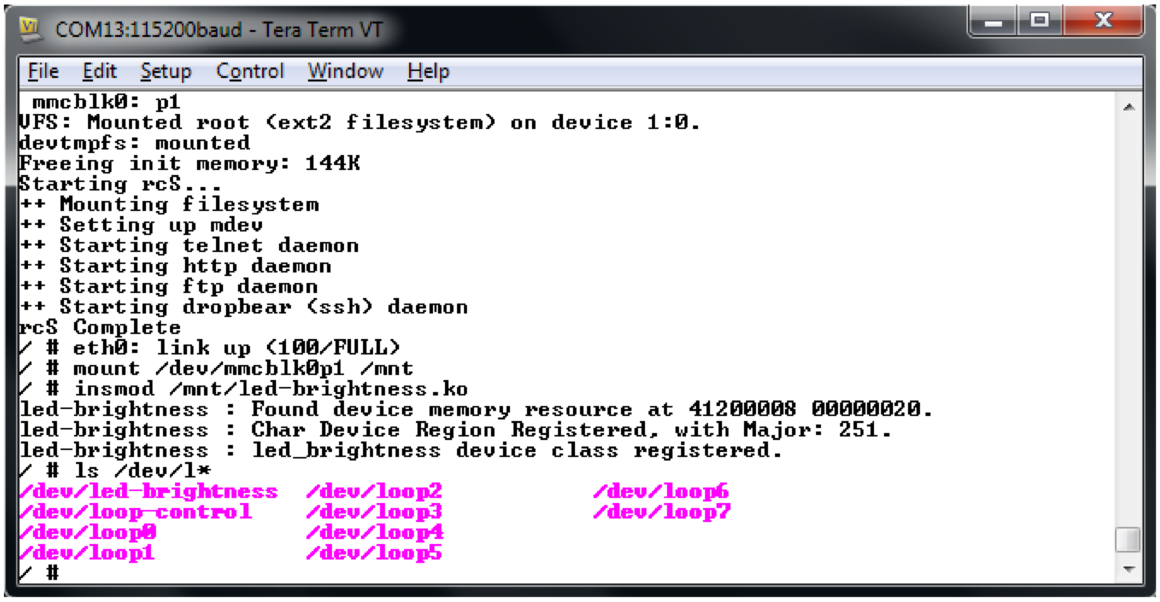
```
IP-Config: Guessing netmask 255.255.255.0
IP-Config: Complete:
     device=eth0, addr=192.168.1.10, mask=255.255.255.0, gw=255.255.255.255,
     host=192.168.1.10, domain=, nis-domain=(none),
     bootserver=255.255.255.255, rootserver=255.255.255.255, rootpath=
RAMDISK: gzip image found at block 0
mmc0: new SD card at address e624
mmcblk0: mmc0:e624 SU02G 1.84 GiB
 mmcblk0: p1
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing init memory: 144K
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
rcS Complete
/ # mount /dev/mmcblk0p1 /mnt/
/ # cd /mnt/
/mnt # insmod led-brightness.ko
/mnt #
```

Figure 37 – Inserting the Driver Module Into the Kernel

10. Take a look in the /dev folder for the led-brightness character device.

```
# ls /dev/l*
```

Notice that this character device is now shown in the /dev folder. The driver is working as we expected and the device is ready for use in the rest of the lab.


COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
 mmcblk0: p1
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing init memory: 144K
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
rcS Complete
/ # eth0: link up (100/FULL)
/ # mount /dev/mmcblk0p1 /mnt
/ # insmod /mnt/led-brightness.ko
led-brightness : Found device memory resource at 41200008 00000020.
led-brightness : Char Device Region Registered, with Major: 251.
led-brightness : led_brightness device class registered.
/ # ls /dev/l*
/dev/led-brightness  /dev/loop2           /dev/loop6
/dev/loop-control    /dev/loop3           /dev/loop7
/dev/loop0           /dev/loop4
/dev/loop1           /dev/loop5
/ #
```

Figure 38 – Device Driver Module Inserted and led-brightness Devices Listed

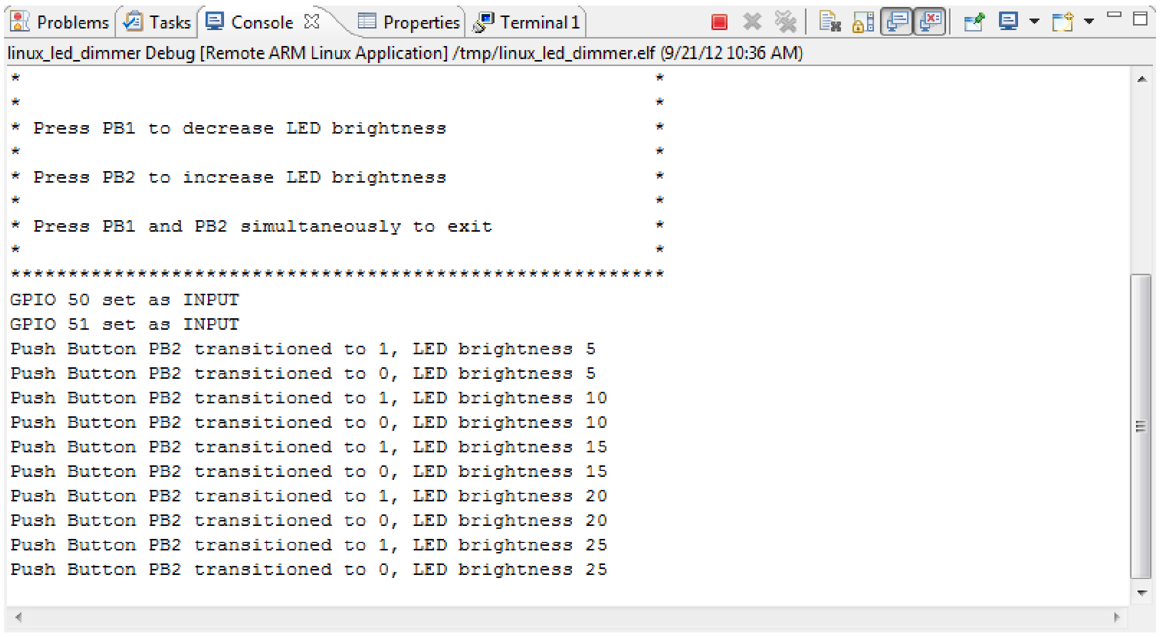
11. Un-mount the SD card from the /mnt mount point.

```
# cd /

# umount /mnt/
```

12. In the SDK **Project Explorer** tab, right-click the **linux_led_dimmer** project and select the **Run As→Run Configurations...** menu item.

    Select the **linux_led_dimmer Debug** configuration from the left and re-launch the application run configuration by clicking on the **Run** button. This time, the console should not display the same device error message.

    Use the PB1 and PB2 buttons as indicated on the display. Use PB1 to decrease the brightness of the LD0-LD7 LEDs and PB2 to increase the brightness of the same LEDs. According to the application, pressing PB1 and PB2 simultaneously should cause the application to exit. Experiment with the use of these buttons to interact with the application. Do you find any inconsistent behavior?



linux_led_dimmer Debug [Remote ARM Linux Application] /tmp/linux_led_dimmer.elf (9/21/12 10:36 AM)

```
*                                                        *
*                                                        *
* Press PB1 to decrease LED brightness                   *
*                                                        *
* Press PB2 to increase LED brightness                   *
*                                                        *
* Press PB1 and PB2 simultaneously to exit               *
*                                                        *
**********************************************************
GPIO 50 set as INPUT
GPIO 51 set as INPUT
Push Button PB2 transitioned to 1, LED brightness 5
Push Button PB2 transitioned to 0, LED brightness 5
Push Button PB2 transitioned to 1, LED brightness 10
Push Button PB2 transitioned to 0, LED brightness 10
Push Button PB2 transitioned to 1, LED brightness 15
Push Button PB2 transitioned to 0, LED brightness 15
Push Button PB2 transitioned to 1, LED brightness 20
Push Button PB2 transitioned to 0, LED brightness 20
Push Button PB2 transitioned to 1, LED brightness 25
Push Button PB2 transitioned to 0, LED brightness 25
```

Figure 39 – Application Output in SDK Console

13. Exit SDK through the **File→Exit** menu item.

Questions:

Answer the following questions:

- *Why is does the led-brightness driver module need to be inserted for the application to run properly?*

____

____

____

Exploring Further

If you have additional time and would like to investigate more…

- Experiment with the SDK environment.

This concludes Lab 4.1.

Revision History

Date	Version	Revision
19 Sep 12	00	Initial Draft
01 Oct 12	01	Revised Draft
19 Oct 12	02	Course Release
14 Jan 13	05	ZedBoard.org Training Course Release

Resources

http://www.zedboard.org

http://www.xilinx.com/zynq

http://www.xilinx.com/planahead

http://www.xilinx.com/sdk

Answers

Experiment 1

- *What communications mechanism is used to command the target ZedBoard from SDK?*

SSH over Ethernet

Experiment 2

- *Why is does the led-brightness driver module need to be inserted for the application to run properly?*

The led-brightness driver module must be present and enumerated in the system because the linux_led_dimmer application uses the device driver to access the custom hardware peripheral in the programmable logic.