Lab 3: Application Development and Debug

Introduction

The PetaLinux tools allow you to easily write your application and build it into the embedded Linux image. In most cases, you write your application on your general (host) computer system instead of the embedded target system on which the embedded Linux runs. In these cases, you will need cross-compilation. The PetaLinux tools allow you to cross-compile the embedded Linux application on the desktop Linux.

The PetaLinux tools support debugging Zynq® All Programmable SoC user applications with the System Debugger using TCF (target communication framework) agent. With TCF you are allowed to debug your application running on the target remotely.

In previous labs, you learned how to build the standard embedded Linux target for an ARM® Cortex™-A9 processor system. While the embedded Linux distribution contains a large number of useful applications and utilities, it is very likely that in order to meet your system requirements you will need to write your own application programs.

The goal of this lab is to help you to create, develop, build, run, and debug your own application on the ARM Cortex-A9 MPcore embedded Linux. The example application will be simple; however the concepts and principles all apply directly to large, complex applications.

This lab builds directly on the skills learned in previous labs, specifically building and booting the Linux system and logging in to the ARM Cortex-A9 MPcore Linux system. Refer to the earlier labs if you have any doubts about these processes or ask your instructor.

Objectives

After completing this lab, you will be able to:

- Create a simple user application with the PetaLinux tools
- Build the new user application by cross-compiling into the embedded Linux
- Run the application in embedded Linux
- Debug the application with the System Debugger

Preparation

If this is the first lab that you are performing, then refer to the “Before You Start” section of Lab 1 for necessary preparatory information on how to set up the environment.

If your workstation has been restarted or logout, run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Please refer to the “Initializing the Workshop Environment” section of Lab 1 for detailed information.

General Flow for this Lab

Creating a New User Application Step 1

To run a user application on an ARM Cortex-A9 MPcore system, you need to cross-compile it and build it into the embedded Linux image on a host machine. PetaLinux tools make these steps easy.

1-1. Change the path to the project directory.

1-1-1. Run the following commands to create and change to the project directory path:

```
[host] $ mkdir ~/emblnx/labs/lab3

[host] $ cd ~/emblnx/labs/lab3
```

1-2. Use the `petalinux-create` command to create a new embedded Linux platform and choose the platform.

1-2-1. Run the following command to create a new petalinux project:

```
[host] $ petalinux-create -t project -s /opt/pkg/ZYBO_petalinux_v2014_2.bsp
```

The command will create the PetaLinux software project directory: `ZYBO_petalinux_v2014_2` under `~/emblnx/labs/lab3`.

1-2-2. Change the directory to the PetaLinux project:

```
~/emblnx/labs/lab3/ZYBO_petalinux_v2014_2
```

1-3. Create a new application.

The PetaLinux tools allows you to create user application templates for either C or C++. These templates include application source code and makefiles so that you can easily configure and compile applications for the target and install them into the root file system.

1-3-1. Enter the following command to create a new user application inside a PetaLinux project:

```
[host] $ petalinux-create -t apps --name myapp --template c
```

```
petalinux@ubuntu:~/emblnx/labs/lab3/ZYBO_petalinux_v2014_2$ petalinux-create -t
apps --name myapp --template c
INFO: Create apps: myapp
INFO: New apps successfully created in /home/petalinux/emblnx/labs/lab3/ZYBO_pet
alinux_v2014_2/components/apps/myapp
```

Figure 1. Creating a new application

The new application you have created can be found in the `<project-root>/components/apps/myapp` directory, where `<project-root>` is `~/emblnx/labs/lab3/ZYBO_petalinux_v2014_2`.

To create a C++ application template, pass the `--template c++` option.

1-3-2. Change to the newly created application directory:

```
[host] $ cd <project-root>/components/apps/myapp
```

You will see the following PetaLinux template-generated files:

Figure 2. Files Generated After Creating a New Application

Template	Description
`Kconfig`	Configuration file template. This file controls how your application is integrated into the PetaLinux menu configuration system and allows you to add configuration options for your own application to control how it is built or installed.
`Makefile`	Compilation file template. This is a basic makefile containing targets to build and install your application into the root file system. This file needs to be modified when you add additional source code files to your project.
`README`	A file to introduce how to build the user application.
`myapp.c` for C or `myapp.cpp` for C++	Simple application program in either C or C++, depending upon your choice. These files will be edited or replaced with the real source code for your application.

Building the Application into the Embedded Linux Step 2

Once you have created the new application, the next step is to compile and build it.

2-1. Select the new application to be included in the build process. The application is not enabled by default.

2-1-1. Ensure that your current working directory (project directory) is `~/emblnx/labs/lab3/ZYBO_petalinux_v2014_2`

2-1-2. Launch the rootfs configuration menu by entering the following command:

```
[host] $ petalinux-config -c rootfs
```

Note: Make sure that the terminal window is at least 80 columns wide.

The linux/rootfs configuration menu opens.

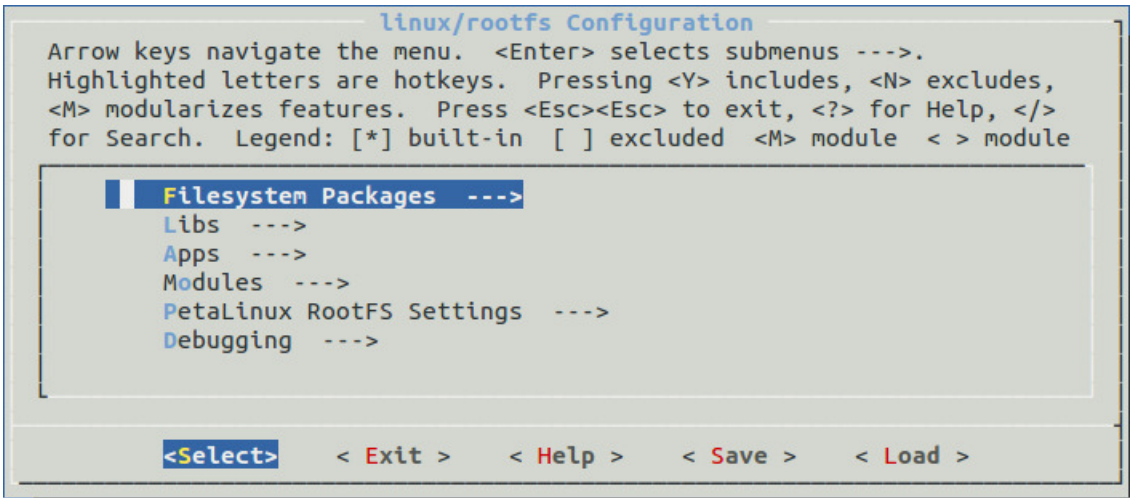

Figure 3. The linux/rootfs Configuration menu

2-1-3. Press the Down Arrow key (↓) to scroll down the menu to **Apps**.

2-1-4. Press <**Enter**> to go into the Apps sub-menu.

The new application **myapp** is listed in the menu.

2-1-5. Move to **myapp** and press <**Y**> to select the application.

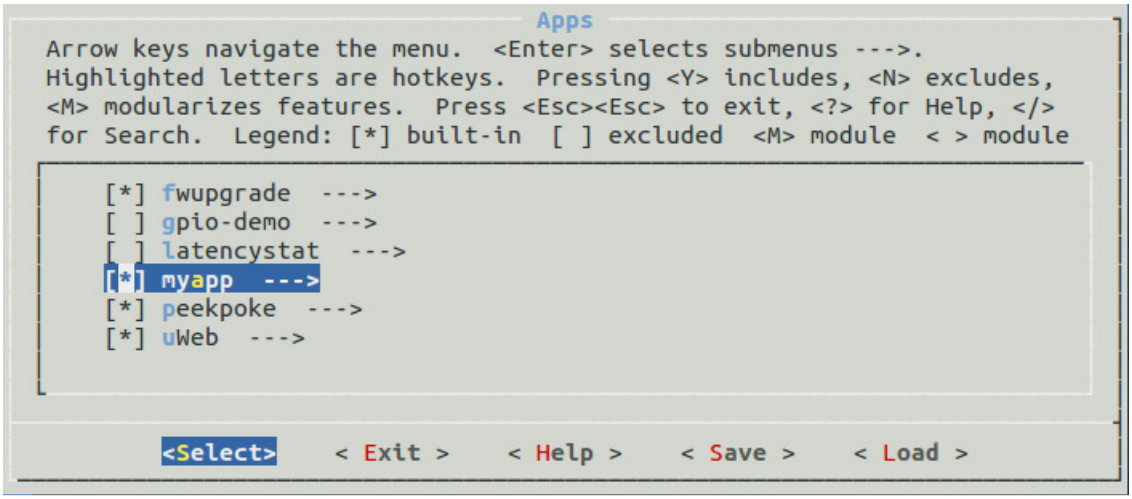

Figure 4. Selecting the new application `myapp`

2-1-6. Press <**Enter**>to open the myapp options.

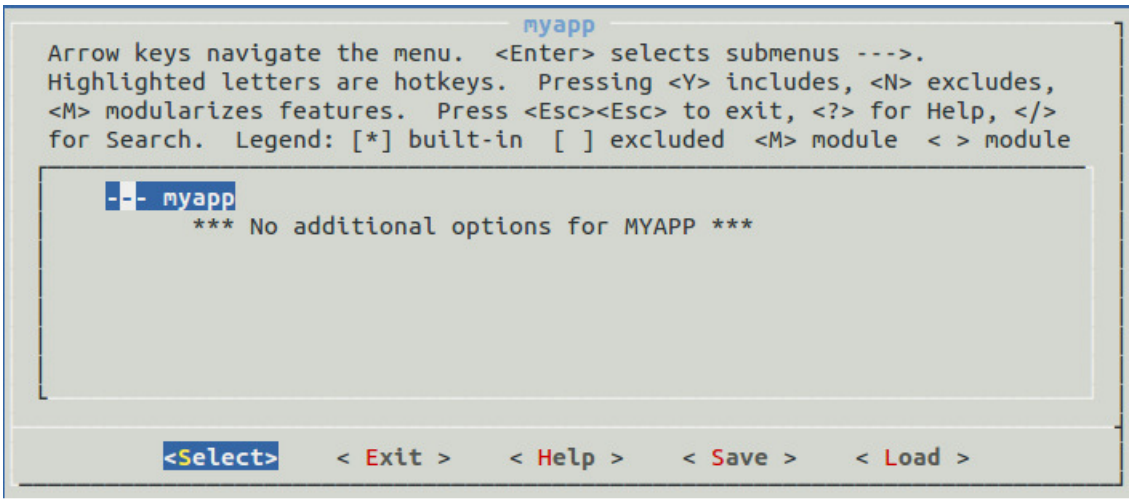

Figure 5. No additional options for `myapp` sub-menu

By default, there are no additional options for `myapp`. Advanced users can modify the `Kconfig` file in the `myapp` directory to add custom options.

2-1-7. Select and press **Exit** several times to return to the main menu.

2-2. Enable the build debug-able applications.

2-2-1. Scroll down the linux/rootfs configuration menu to **Debugging**.

2-2-2. Select the **Debugging** sub-menu and ensure that `build debugable applications` is selected. Click **Y** to select `build debugable applications`.

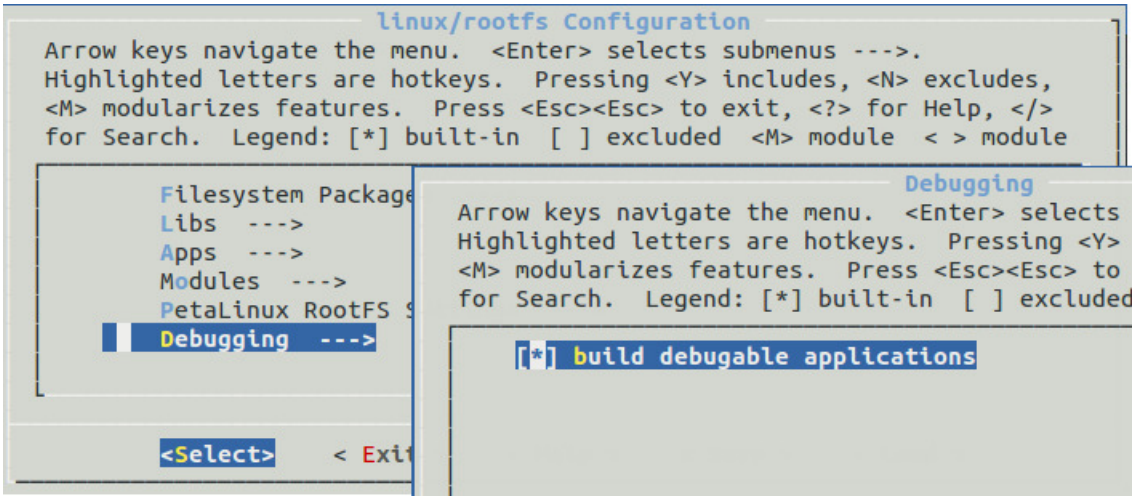

Figure 6. Selecting build debug-able applications

2-2-3. Select and press **Exit** to return to the main menu.

2-3. Enable the TCF agent for debugging support. The PetaLinux tools support debugging Zynq All Programmable SoC user applications with a TCF agent.

2-3-1. Select **Filesystem Packages**.

2-3-2. In the Filesystem Packages menu, select **base**.

2-3-3. In the base menu, scroll down and select **tcf-agent**.

2-3-4. Enable the **tcf-agent** by clicking **Y**.

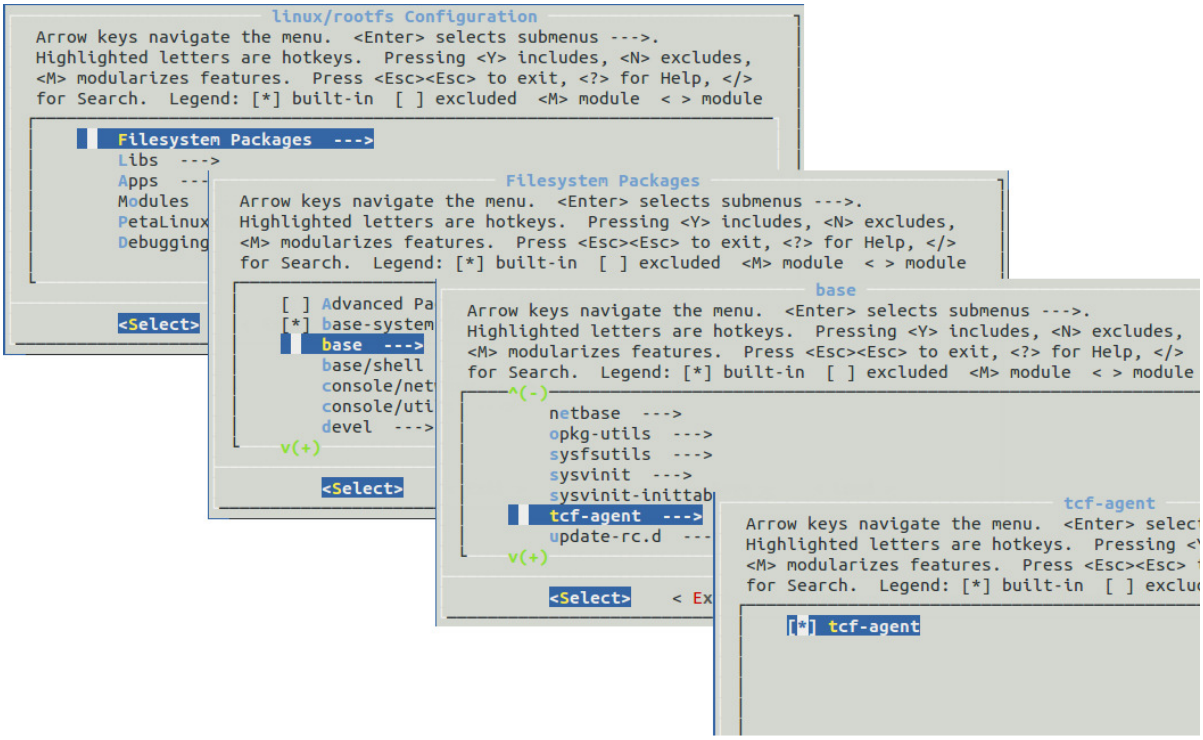

Figure 7. Enabling the TCF Agent

2-3-5. Select and press **Exit** to return to the main menu.

2-3-6. Exit the linux/rootfs configuration menu.

2-3-7. Select <**Yes**> to save your new configuration.

It will take a few seconds for the configuration changes to be applied. Wait until you return to the shell prompt on the command console.

2-4. Build the image.

2-4-1. Enter the following command to build the image:

```
[host] $ petalinux-build
```

Booting the New Image with QEMU Step 3

With the QEMU simulator, you can develop and debug the software application without using hardware at all.

3-1. Run the application in QEMU.

3-1-1. Enter the following command to boot the newly built PetaLinux image through QEMU:

```
# petalinux-boot --qemu --kernel
```

3-1-2. After the system boots, log into the system by entering `root` as the both the login name and password.

3-1-3. Examine the `/bin` directory in the QEMU console:

```
# ls /bin | grep myapp
```

You should see that the `myapp` application is there.

3-1-4. Execute the `myapp` application in the QEMU console:

```
# myapp 1
```

The following is the output of the command:

```
Hello, PetaLinux World!
cmdline args:
myapp
1
```

3-1-5. Press <**Ctrl + a**> then press <**x**> to exit QEMU.

Debugging the Application Using System Debugger in Board Step 4

4-1. Launch XSDK and create a workspace.

4-1-1. Open a new terminal.

4-1-2. Enter the following command to create the `workspace` directory under `lab3`:

```
[host] $ mkdir ~/emblnx/labs/lab3/workspace
```

4-1-3. Enter the following command to launch xsdk:

```
[host] $ xsdk
```

If the program does not start then source the settings by executing the following command:

```
[host] $ source /opt/pkg/Xilinx/Vivado/2014.2/settings32.sh
```

4-1-4. Select `~/emblnx/labs/lab3/workspace` as the workspace directory.

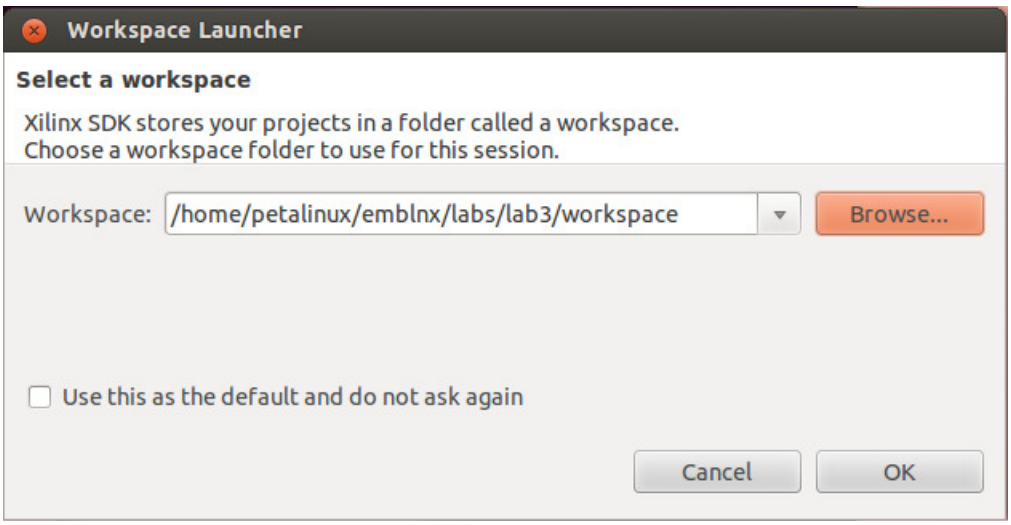

Figure 8. Setting pp the Workspace Environment path

4-1-5. Click **OK**.

4-1-6. Close the *Welcome* screen if it is open.

4-2. Create the hardware platform specification.

4-2-1. Select **File** > **New** > **Project**.

4-2-2. In the pop-up window select **Xilinx** > **Hardware Platform Specification**.

4-2-3. Click **Next**.

4-2-4. Enter **zynq_hw_platform** as the project name.

4-2-5. Under the Target Hardware Specification region, browse to the `~/emblnx/sources/lab3/` directory and select **system_wrapper.hdf**.

4-2-6. Click **OK**.

4-2-7. Click **Finish**.

4-3. Make sure that the BOOT.BIN file located in the SD card is copied from the pre-built directory.

4-3-1. Make sure that the pre-built `BOOT.BIN` file is located in the SD card.

If you have done Lab 2 as your last lab, there is no need to make any changes to the SD card.

If not, copy the `BOOT.BIN` from the `~/emblnx/lab3/ZYBO_petalinux_v2014_2/pre-built/linux/images` directory to the SD card.

4-3-2. Insert the SD card back to the target board.

4-4. Run the DHCP server on the host.

4-4-1. Run the DHCP server:

```
[host]$ sudo service isc-dhcp-server restart
```

4-5. Power up the board and set the serial port terminal.

4-5-1. Power ON the board.

4-5-2. Make sure that `/dev/ttyUSB1` is set to read/write access:

```
[host]$ sudo chmod 666 /dev/ttyUSB1
```

4-5-3. In the dashboard, in the Search field, enter the serial port.

4-5-4. Select the **Serial port terminal** application.

You can reset (BTN7) the board to see the booting info once again.

4-6. Boot the new Linux image on the board.

4-6-1. Watch the booting process in the GtkTerm window.

4-6-2. Press any key to stop auto-boot when you see messages similar to the following in the GtkTerm window:

```
*****************************************************************
*            ZYBO Unique MAC Addr: 00 04 A3 D1 D5 B2            *
*---------------------------------------------------------------*
* The Unique MAC address printed above can be used by modifying *
* the u-boot environment stored in QSPI flash. This is done with *
* the saveenv command in u-boot. Refer to the Xilinx document   *
* titled "Petalinux Tools User Guide: Firmware Upgrade Guide"   *
* (UG983) for more information.                                 *
*****************************************************************


U-Boot 2014.01 (Jul 01 2014 - 19:56:17)

Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL128S_64K with page size 256 Bytes, erase size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
U-BOOT for ZYBO_petalinux_v2014_2

Gem.e000b000 Waiting for PHY auto negotiation to complete........ done
BOOTP broadcast 1
DHCP client bound to address 192.168.1.11
Hit any key to stop autoboot:  0
```

Figure 9. Stopping the Autoboot

4-6-3. If you did not see the "DHCP client bound to address" message during the uboot bootup, you will need to run `dhcp` to obtain the IP address:

```
U-Boot-PetaLinux> dhcp
```

4-6-4. Set the TFTP server IP to the host IP by running the following command in the u-boot console:

```
U-Boot-PetaLinux> set serverip 192.168.1.1
```

4-6-5. Download and boot the new image using TFTP by executing this command in the u-boot console:

```
U-Boot-PetaLinux> run netboot
```

This command will download the `image.ub` file from `/tftpboot` on the host to the main memory of the ARM Cortex-A9 MPcore system and boot the system with the image.

4-6-6. After the system boots, log into the system by entering `root` as both the login name and password.

4-6-7. Confirm that TCF agent is running.

4-6-8. Enter the following command in the GtkTerm window to verify the IP address of the board:

```
# ifconfig
```

4-6-9. Verify the eth0 IP address.

4-7. Create a new Debug configuration and configure the setup of the target.

4-7-1. From SDK, select **Run** > **Debug Configurations…**

Make sure that the SDK window is active as the selected foreground window.

4-7-2. Double-click **Xilinx C/C++ application (System Debugger)** to create a new configuration.

4-7-3. Select **Linux Application Debug** as the debug type.

4-7-4. Set the host name to the IP address of the target board as determined in the previous step.

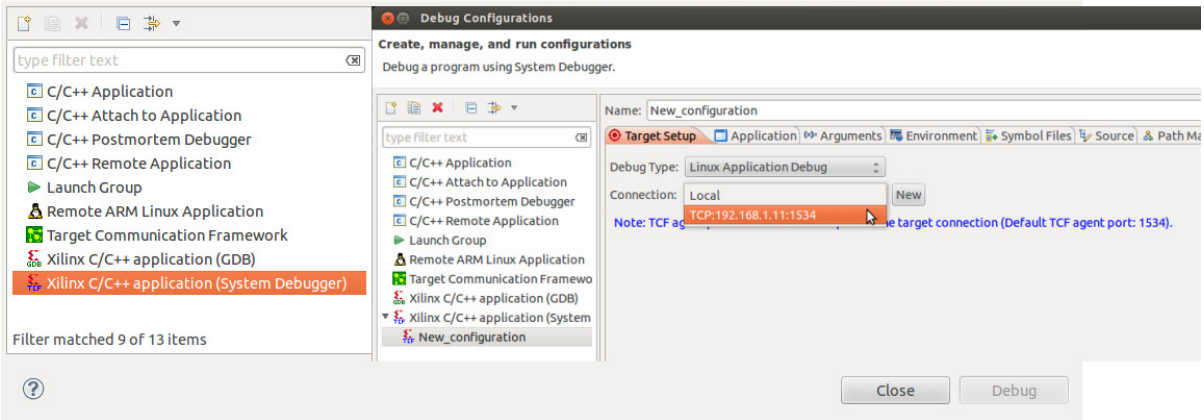

Figure 10. Configuring the Target Setup

4-8. Configure both the local and remote file paths.

4-8-1. Select the **Application** tab.

4-8-2. Set the local file path to be the compiled application in the project directory by clicking **Browse** and locating the following folder:

```
<project-root>/build/linux/rootfs/apps/myapp/myapp
```

4-8-3. Set the remote file path to be the location on the target file system where your application can be found:

```
/bin/myapp
```

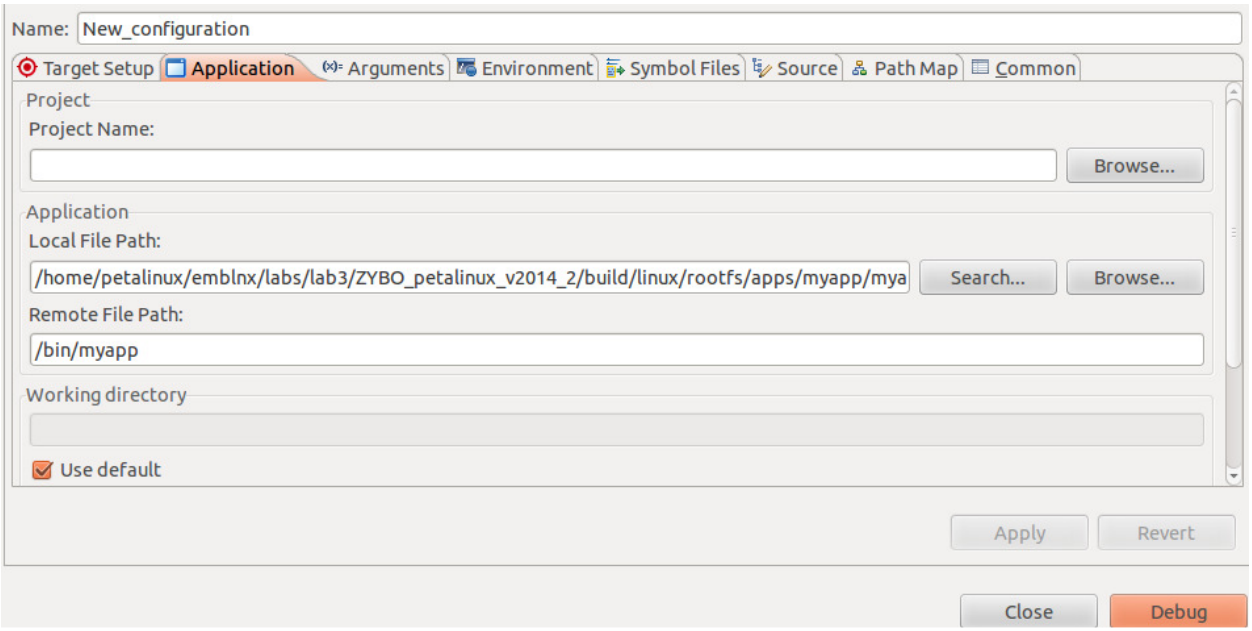

Figure 11. Configuring the Application File paths

4-8-4. Click **Apply**.

4-9. Debug the program.

4-9-1. Click **Debug**.

4-9-2. Click **Yes** to confirm the perspective switch.

The Debug perspective opens.

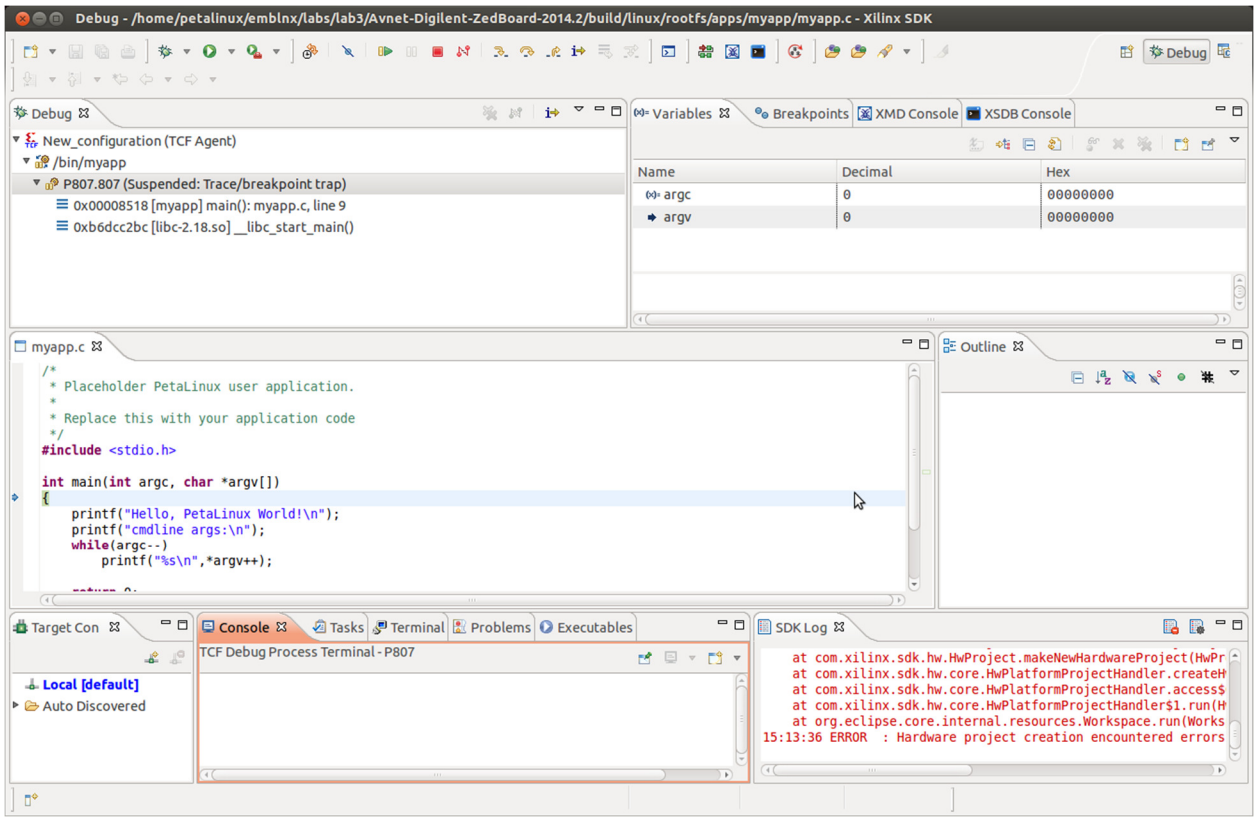

Figure 12. Debug perspective window

Program operation is suspended at the first executable statement in *main{}* (not running).

Note that local variables for the current function are shown in the Variables tab.

4-9-3. Select **Window** > **Show View** > **Disassembly**.

4-9-4. Double-click line number **11** to set a breakpoint there (a check mark becomes visible).

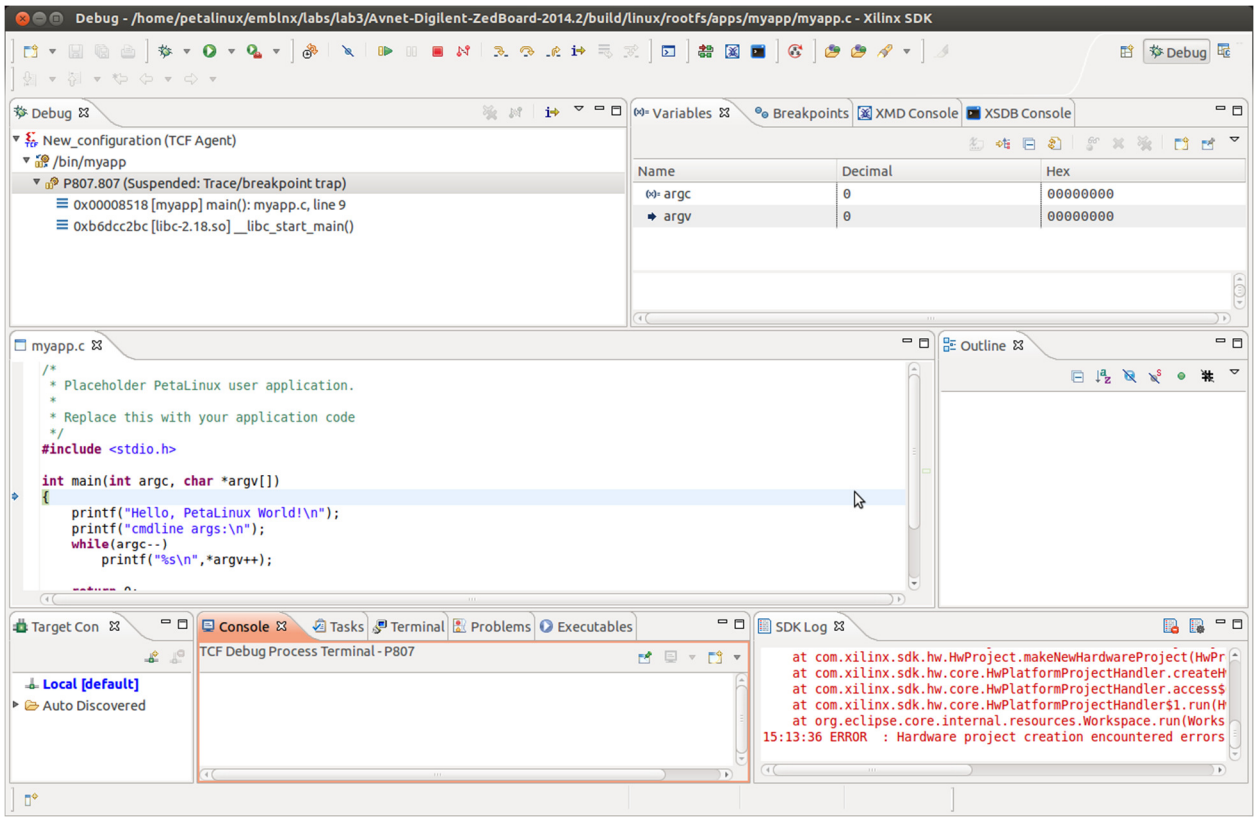

Figure 13. Breakpoint set at line number 11

Note: If line numbers are not displayed, right-click the leftmost column of the editor and select **Show Line Numbers**.

4-10. Resume the program.

4-10-1. Click the **Play**/**Resume** button (green triangle) to run the program.

The program runs until the breakpoint.

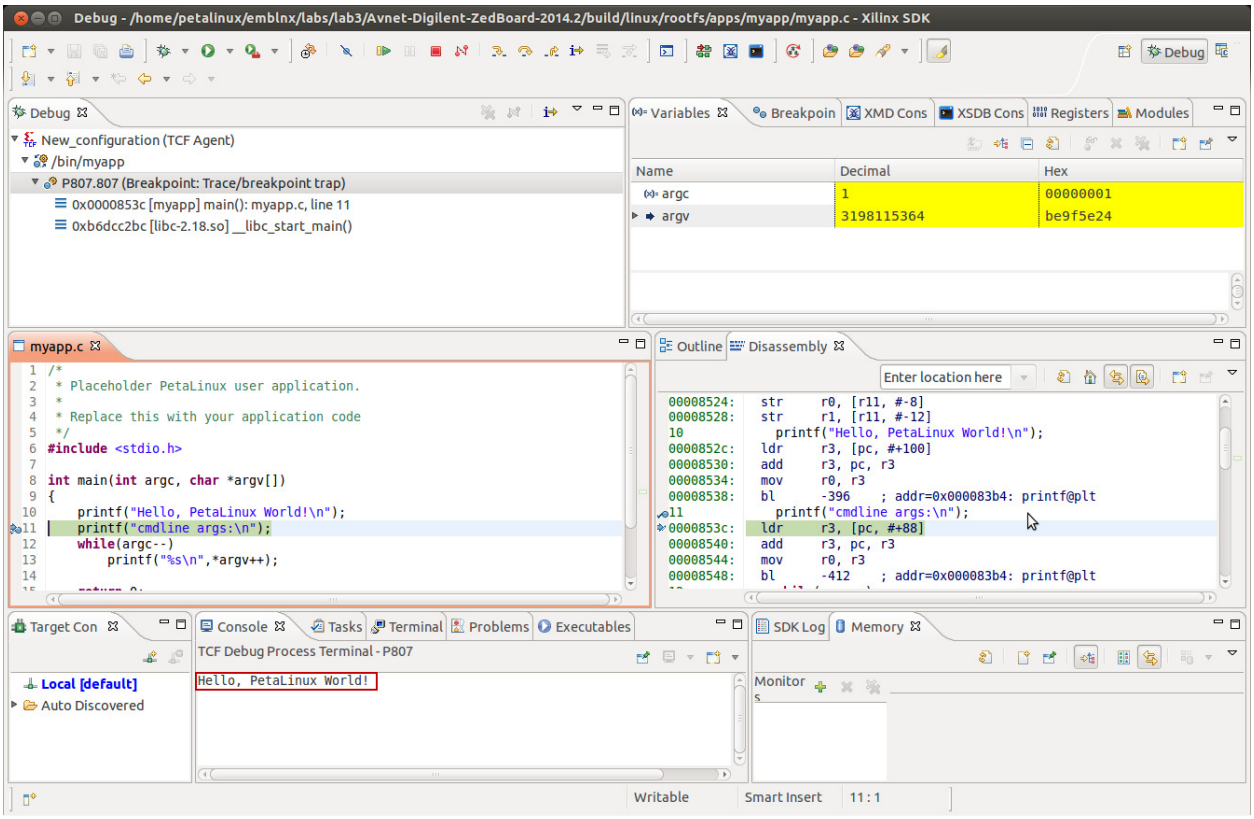

Figure 14. Program Stops at breakpoint

You will see the message in the Console window.

You can explore other options and disconnect the program by clicking the **Disconnect** ( ) button.

4-10-2. Close the XSDK tool by selecting **File > Exit**.

Customizing the Application Template Step 5

5-1. Edit the source file to print a configurable welcome string.

5-1-1. In the host terminal, change the directory to:

```
[host]$cd
~/emblnx/labs/lab3/ZYBO_petalinux_v2014_2/components/apps/myapp
```

5-1-2. Enter the following command to edit the `myapp.c` file:

```
[host]$gedit myapp.c
```

5-1-3. Add the lines that are outlined in the figure below.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        char *welcome;
#ifdef WELCOME
        welcome=WELCOME;
#else
        welcome="PetaLinux World!";
#endif
        printf("Hello, %s\n",welcome );
        printf("cmdline args:\n");
        while(argc--)
                printf("%s\n",*argv++);

        return 0;
}
```

Figure 15. Customizing the myapp file

Note: Observe that the first *printf* statement needs to be modified to use the corresponding value of *welcome*.

5-1-4. Save and close the file.

5-2. Edit the `Kconfig` file of the user application to add a configuration option.

5-2-1. Enter the following command to edit the `Kconfig` file:

```
[host]$gedit Kconfig
```

5-2-2. Add the lines that are outlined in the figure below.

```
if ROOTFS_COMPONENT_APPS_NAME_MYAPP
        comment "No additional options for MYAPP"

        config APPS_MYAPP_WELCOME
        string "Welcome String"
        help
        Welcome string for myapp

#       config APPS_MYAPP_OPTION0
#       bool "option0"
#       help
#         Help text
endif
```

Figure 16. Adding a Configuration Option in the Kconfig file

5-2-3. Save and close the file.

5-3. Edit the user application makefile to pass the configurable option to the user application executable.

5-3-1. Enter the following command to edit the makefile:

```
[host]$gedit Makefile
```

5-3-2. Add the lines that are outlined in the figure below.

```
include apps.common.mk

include $(ROOTFS_CONFIG)

ifneq ($(CONFIG_APPS_MYAPP_WELCOME),)
CFLAGS += -DWELCOME=\"$(CONFIG_APPS_MYAPP_WELCOME)\"
endif

APP = myapp
```

Figure 17. Modifying the Makefile

5-3-3. Save and close the file.

5-4. Re-run the application configuration menu.

5-4-1. Change to the `<project-root>` directory. That is, change to `~/emblnx/labs/lab3/ZYBO_petalinux_v2014_2`

5-4-2. Enter the following command:

```
[host]$petalinux-config -c rootfs
```

5-4-3. Select **Apps** > **myapp**.

You should see the new Welcome String configuration in the myapp sub-menu.

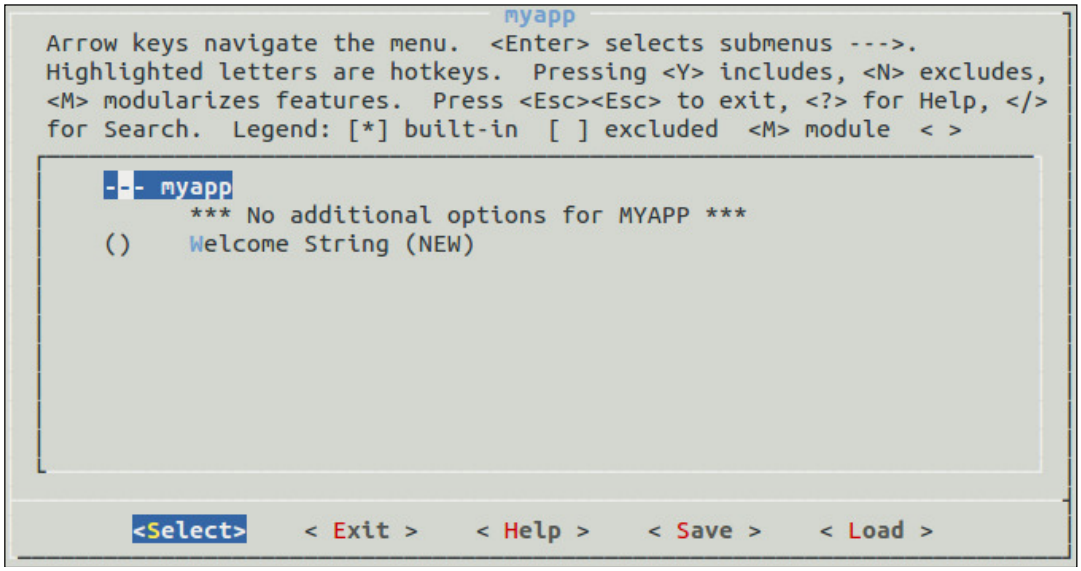

Figure 18. Welcome string configuration

5-4-4. Select the **Welcome String** option.

5-4-5. Enter **It's a user application test!**.

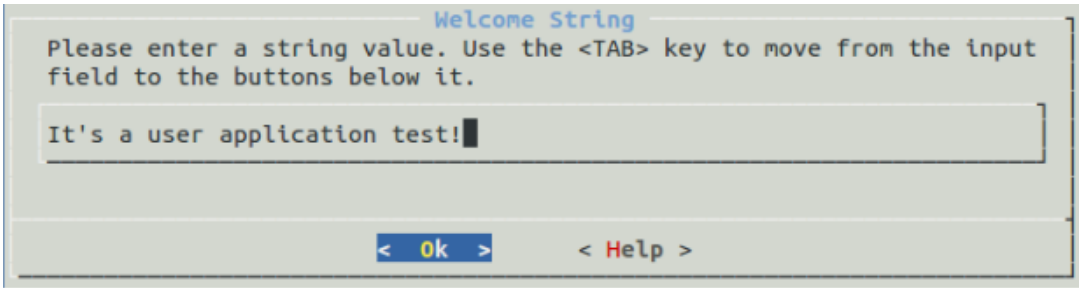

Figure 19. Entering the Welcome string

5-4-6. Exit and save the configuration change.

5-5. Rebuild the image.

5-5-1. Change to the `<project-root>` directory.

That is, change to `~/emblnx/labs/lab3/ZYBO_petalinux_v2014_2`

5-5-2. Rebuild application and target the system image:

```
[host]$ petalinux-build -c rootfs/myapp -x clean
```

```
[host]$ petalinux-build -c rootfs/myapp
[host]$ petalinux-build -x package
```

5-6. Run the application in QEMU.

5-6-1. Enter the following command to boot the newly built PetaLinux image through QEMU:

```
[host]$ petalinux-boot --qemu --kernel
```

5-6-2. After the system boots, log into the system.

5-6-3. Execute the `myapp` application in the QEMU console:

```
# myapp
```

The following is the output of the command:

```
Hello, It's a user application test!
cmdline args:
myapp
```

5-6-4. Press <**Ctrl + a**> then press <**x**> to exit QEMU.

Conclusion

In this lab, you have learned how to:

- Create an ARM Cortex-A9 MPcore embedded Linux application
- Build your application with cross-compilation
- Run the application in QEMU
- Debug your application by using System Debugger

Note that although System Debugger can be used to debug your application, printing or logging information whenever necessary in your application is very important for tracking issues.

Completed Solution

If you want to run the solution then copy **`BOOT.bin`** from the *`labsolution\lab3\SDCard`* directory onto a SD card. Place the SD card in the Zybo. Set Zybo in the SD Card boot mode. Connect the Zybo to the host machine using Ethernet cable.

Run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Copy the **`image.ub`** file from the *`labsolution\lab3\tftpboot`* directory into */tftpboot* directory.

Power ON the board. Set the terminal session. Interrupt the boot process when autoboot message is shown. Set the serverip address using the following command in the target board terminal window:

```
#set serverip 192.168.1.1
```

Run the netboot command:

```
#run netboot
```

Login into the system and test the lab.