PetaLinux SDK User Guide

Application Development Guide

UG981 (v2013.10) November 25, 2013

XILINX®

Revision History

Date	Version	Notes
2010-03-09	1.1	Initial public release
2010-11-10	1.2	The document has been updated. In PetaLinux SDK 1.2, users don't have to manually set the shared library search path for the GDB client.
2010-11-26	1.3	Updated to refer to PowerPC targets
2011-04-04	2.1	Updated for PetaLinux SDK 2.1 release
2011-10-18	2.2	Updated for PetaLinux SDK 2.2 release
2012-08-03	3.1	Updated for PetaLinux SDK 3.1 release
2012-09-03	12.9	Updated for PetaLinux SDK 12.9 release
2012-12-17	2012.12	Updated for PetaLinux SDK 2012.12 release
2013-04-29	2013.04	Updated for PetaLinux SDK 2013.04 release
2013-11-25	2013.10	Updated for PetaLinux SDK 2013.10 release

Online Updates

Please refer to the PetaLinux v2013.10 Master Answer Record (Xilinx Answer Record #55776) for the latest updates on PetaLinux SDK usage and documentation.

XILINX

Table of Contents

About this Guide

PetaLinux provides an easy way to develop user applications for Zynq and MicroBlaze Linux systems, including building, installing, and debugging. This guide describes how to create and debug these applications.

IMPORTANT: *It is assumed that the readers of this document have basic Linux knowledge such as how to run Linux commands, and basic PetaLinux familiarity having read and worked through the examples in the Getting Started with PetaLinux SDK Guide.*

Prerequisites

This document assumes that the following prerequisites have been satisfied:

- PetaLinux SDK has been installed.
- Xilinx Software Development Kit (XSDK) has been installed.
- PetaLinux setup script has been sourced in each command console in which you work with PetaLinux. Run the following command to check whether the PetaLinux environment has been setup on the command console:

```
$ echo $PETALINUX
```

- If the PetaLinux working environment has been setup, it should show the path to the installed PetaLinux. If it shows nothing, please refer to section Environment Setup in the Getting Started with PetaLinux SDK document to setup the environment.

New Application

Create New Application

PetaLinux provides a tool 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.

The basic steps are as follows:

1. Create a user application by running `petalinux-create -t apps` from inside a PetaLinux project on your workstation:

   ```
   $ cd <project-root>
   $ petalinux-create -t apps [--template TYPE] --name <user-application-name>
   ```

   e.g., to create a user application called myapp in C (the default):

   ```
   $ petalinux-create -t apps --name myapp
   ```

   or:

   ```
   $ petalinux-create -t apps --template c --name myapp
   ```

   To create a C++ application template, pass the `--template c++` option, as follows:

   ```
   $ petalinux-create -t apps --template c++ --name myapp
   ```

   To create a autoconf application template, pass the `--template autoconf` option, as follows:

   ```
   $ petalinux-create -t apps --template autoconf --name myapp
   ```

   To create an application template which will allow you to copy existing content to the target filesystem, pass the `--template install` option, as follows:

   ```
   $ petalinux-create -t apps --template install --name myapp
   ```

   

   IMPORTANT: *You need to ensure that the binary data being installed into the target file system by an install template application is compatible with the underlying hardware implementation of your system.*

   You can use `-h` or `--help` to see the usage of the `petalinux-create -t apps`. The new application you have created can be found in the `"<project-root>/components/apps/myapp"` directory.

2. Change to the newly created application directory

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

You will see the following PetaLinux template-generated files:

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 filesystem. 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; `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 New Application

Once you have created the new application, the next step is to compile and build it. The required steps are shown below:

1. Select your new application to be included in the build process. The application is not enabled by default. Launch the rootfs configuration menu:

```
$ petalinux-config -c rootfs
```

`linux/rootfs Configuration` menu will show as:

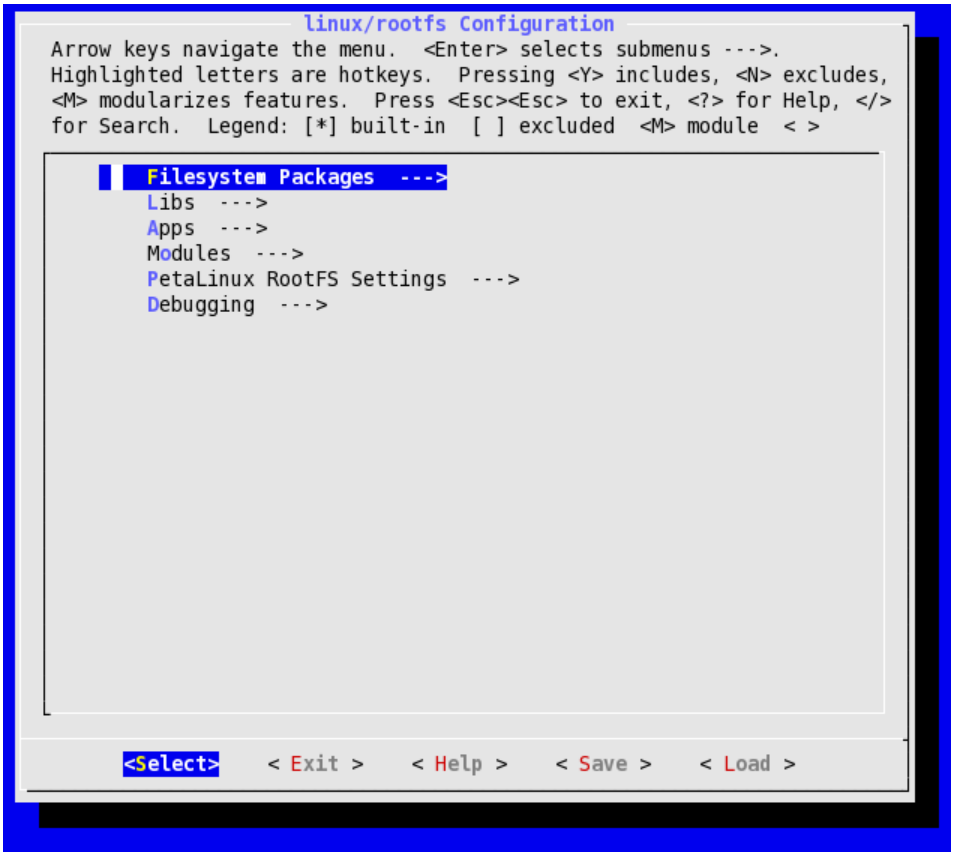

Figure 1: linux/rootfs Configuration Menu

TIP:

- *v(+) below the menu means there are more options below; you can scroll down the menu to see those options by pressing arrow key (↓).*
- *^(-) above the menu means there are more options above; you can scroll up the menu to see those options by pressing arrow key (↑).*

---

2. Press the down arrow key (↓) to scroll down the menu to `Apps`.
3. Press `Enter` to go into the `Apps` sub-menu:

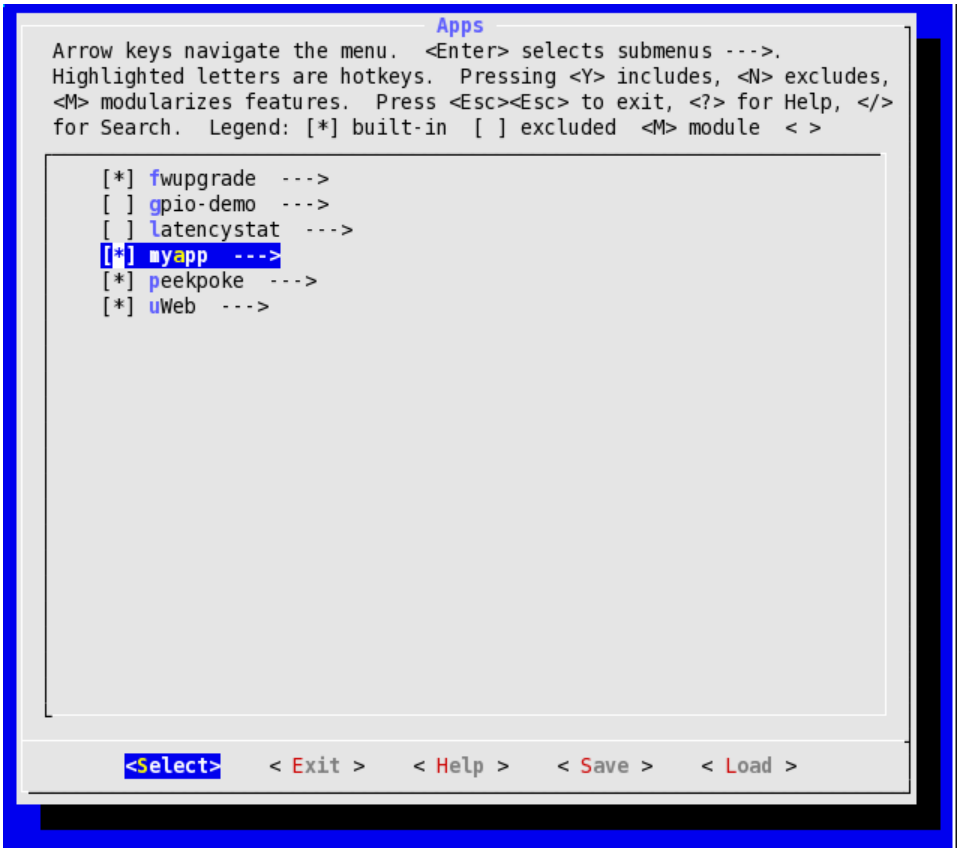

Figure 2: Apps Menu

You will see your newly created user application - `myapp` - listed in the menu.

4. Select and enter the `myapp` sub-menu, which will then show:

Figure 3: 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.

5. Exit the menu and select <Yes> to `Do you wish to save your new configuration?`:

Figure 4: Save new configuration dialog

It will take a few seconds to apply the configuration change, please wait until you return to the shell prompt on the command console.

6. Running `petalinux-build` in the project directory "`<project-root>`" will rebuild the system image including the selected user application - `myapp`. (The output directory for this build process is "`<project-root>/build/linux/rootfs/apps/myapp`")

```
$ petalinux-build
```

To build myapp into an existing system image:

```
$ cd <project-root>
$ petalinux-build -c rootfs/myapp
$ petalinux-build -x package
```

TIP: *Other `petalinux-build` options are explained with `--help`. Some of the build options are:*

- *to clean the selected user application:*

```
$ petalinux-build -c rootfs/myapp -x clean
```

- *to rebuild the selected user application:*

```
$ petalinux-build -c rootfs/myapp -x build
```

- *to install the selected user application:*

```
$ petalinux-build -c rootfs/myapp -x install
```

Testing New Application

So far, you have created and compiled a new application. Now it's time to test it.

1. Boot the newly created system image, either in QEMU or on your hardware board. If you are not sure how to do so, please refer to section `Test New Software Image with QEMU` and section `Test New Software Image on Hardware` in document `Getting Started with PetaLinux SDK`.
2. Confirm your user application is present on the PetaLinux system, by running the following command on the target system login console:

   ```
   # ls /bin
   ```

   Unless you have changed the user application's Makefile to put the user application to somewhere else, the user application will be put in to `"/bin"` directory.
3. Run your user application on the target system console, e.g., to run user application `myapp`:

   ```
   # myapp
   ```

4. Confirm the result of the application is as expected.

If the new application is missing from the target filesystem, double check to make sure that you completed the `petalinux-build -x package` step described in the previous section. This ensures that your application binary is copied into the root filesystem staging area, and that the target system image is updated with this new filesystem.

Debugging MicroBlaze Applications with GDB

PetaLinux supports debugging MicroBlaze user applications with GDB. This section describes the basic debugging procedure.

Preparing the build system for debugging

1. Change the user application Makefile so that compiler optimisations are disabled. Compiler optimisations make debugging difficult because the compiler can re-order or remove instructions that do not impact the program result, making debugging difficult.

   Here is an example taken from a changed Makefile:

   ```
   ...
   include apps.common.mk

   CFLAGS += -O0

   APP = myapp
   ...
   ```

2. Change to the project directory:

   ```
   $ cd <project-root>
   ```

3. Run `petalinux-config -c rootfs` on the command console:

   ```
   $ petalinux-config -c rootfs
   ```

4. Scroll down the linux/rootfs Configuration menu to **Debugging**:

Figure 5: linux/rootfs Configuration - Scrolling down to Debugging option

5. Select the **Debugging** sub-menu and ensure that `build debugable applications` is selected:

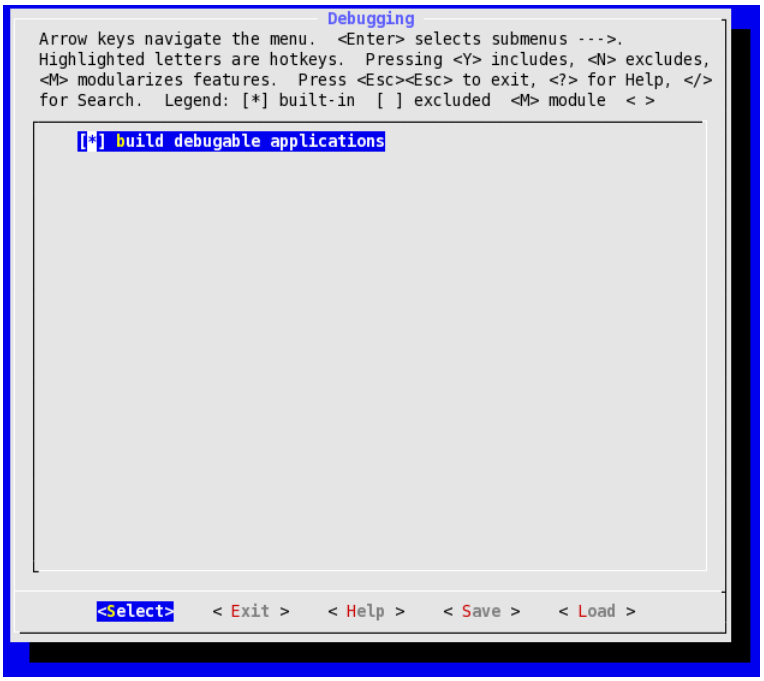

Figure 6: Debugging Menu

6. Exit the Debugging sub-menu, and select the `Filesystem Packages`, followed by the `base` sub-menu:

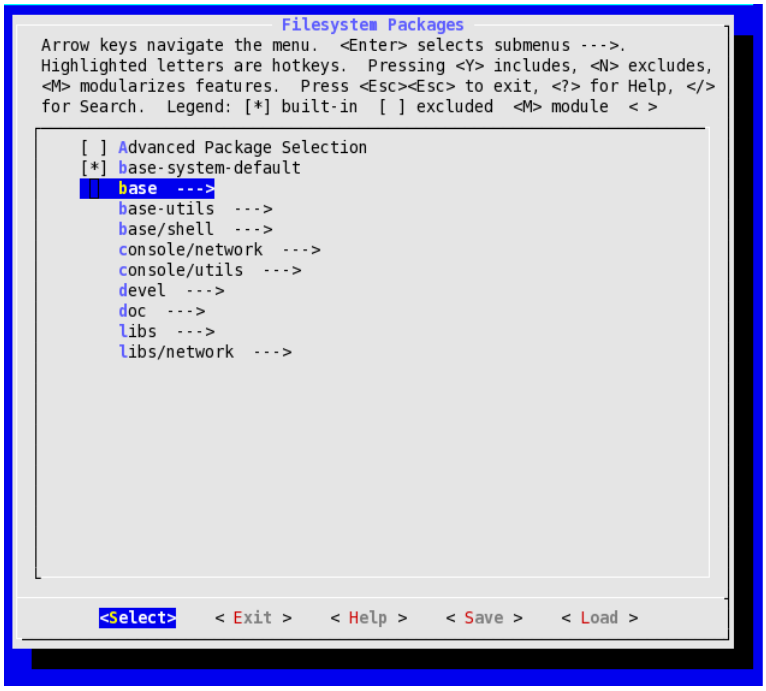

Figure 7: Filesystem Packages Menu

7. Select the `external-xilinx-toolchain` sub-menu option, and ensure `gdbserver` is enabled:

Figure 8: external-xilinx-toolchain sub-menu

8. Exit the menu and select <Yes> to save the configuration.
9. Rebuild the target system image as described previously.

Performing a Debug Session

1. Boot your board (or QEMU) with the new image created above.
2. Run `gdbserver` with the user application on the target system console (set to listening on port 1534):

```
root@Xilinx-KC705-AXI-full-2013.3:~# gdbserver host:1534 /bin/myapp
Process /bin/myapp created; pid = 73
Listening on port 1534
```

   1534 is the `gdbserver` port - it can be any unused port number

3. On the workstation, navigate to the compiled user application's directory:

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

4. Run GDB client

```
$ petalinux-util --gdb myapp
```

5. The GDB console will start:

```
...
GNU gdb (crosstool-NG 1.18.0) 7.6.0.20130721-cvs
...
(gdb)
```

6. In the GDB console connect to the target machine using the command:
   - Use the IP address of the target system, e.g.: `192.168.0.10`. If you are not sure about the IP address, run `ifconfig` on the target console to check.
   - Use the port 1534. If you chose a different gdbserver port number in the earlier step, use that value instead.

IMPORTANT: *If debugging on QEMU, refer to the PetaLinux SDK QEMU System Simulation Guide (UG982) for information regarding IP and port redirection when testing in non-`root` (default) or `root` mode. E.g. if testing in non-root mode, you will need to use `localhost` as the target IP in the subsequent steps.*

```
(gdb) target remote 192.168.0.10:1534
```

The GDB console will attach to the remote target. Gdbserver on the target console will display the following confirmation, where the host IP is displayed;

```
Remote Debugging from host 192.168.0.9
```

7. Before starting the execution of the program create some breakpoints. Using the GDB console you can create breakpoints throughout your code using function names and line numbers. For example create a breakpoint for the `main` function:

```
(gdb) break main
Breakpoint 1 at 0x10000444: file myapp.c, line 10.
```

8. Run the program by executing the `continue` command in the GDB console. GDB will begin the execution of the program and break at any breakpoints.

```
(gdb) continue
Continuing.

Breakpoint 1, main (argc=1, argv=0xbffffe64) at myapp.c:10
10              printf("Hello, PetaLinux World!\n");
```

9. To print out a listing of the code at current program location use the `list` command.

```
(gdb) list
5        */
6       #include <stdio.h>
7
8       int main(int argc, char *argv[])
9       {
10              printf("Hello, PetaLinux World!\n");
11              printf("cmdline args:\n");
12              while(argc--)
13                      printf("%s\n",*argv++);
14
```

10. Try the `step`, `next` and `continue` commands. Try setting and removing breakpoints using the `break` command. More information on the commands can be obtained using the GDB console `help` command.

11. When the program finishes, the GDB server application on the target system will exit. Here is an example of messages shown on the console:

```
Hello, PetaLinux World!
cmdline args:
/bin/myapp

Child exited with status 0
GDBserver exiting
root@Xilinx-KC705-AXI-full-2013.3:~#
```

TIP: *A `.gdbinit` file will be automatically created, to setup paths to libraries. You may add your own GDB initialisation commands at the end of this file if desired.*

Going Further With GDB

For more information on general usage of GDB, please refer to the GDB project documentation:

- http://www.gnu.org/software/gdb/documentation

Debugging Zynq Applications with TCF Agent

PetaLinux supports debugging Zynq user applications with TCF Agent. This section describes the basic debugging procedure.

Preparing the build system for debugging

1. Change to the project directory:

```
$ cd <project-root>
```

2. Run `petalinux-config -c rootfs` on the command console:

```
$ petalinux-config -c rootfs
```

3. Scroll down the linux/rootfs Configuration menu to `Filesystem Packages`, followed by the `base` sub-menu:

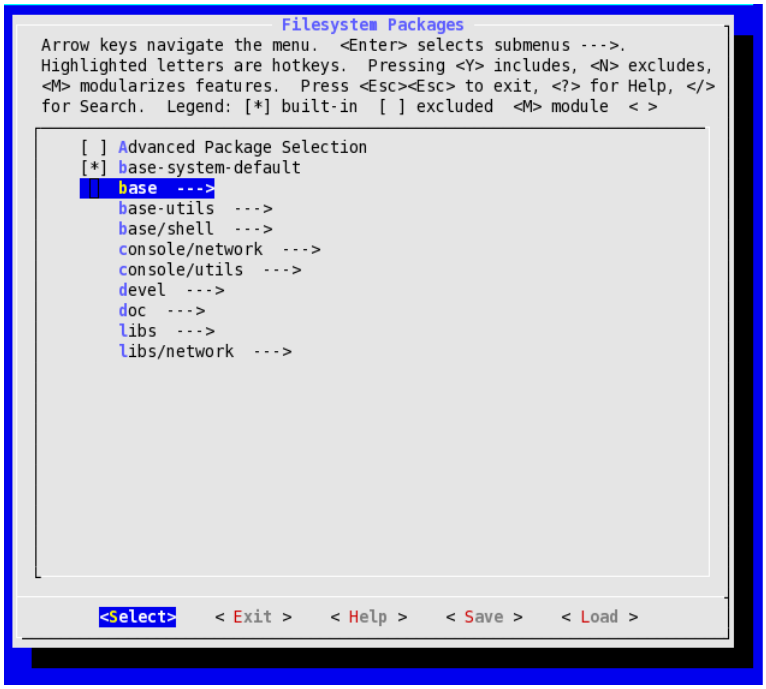

Figure 9: Filesystem Packages Menu

4. Ensure `tcf-agent` is enabled:

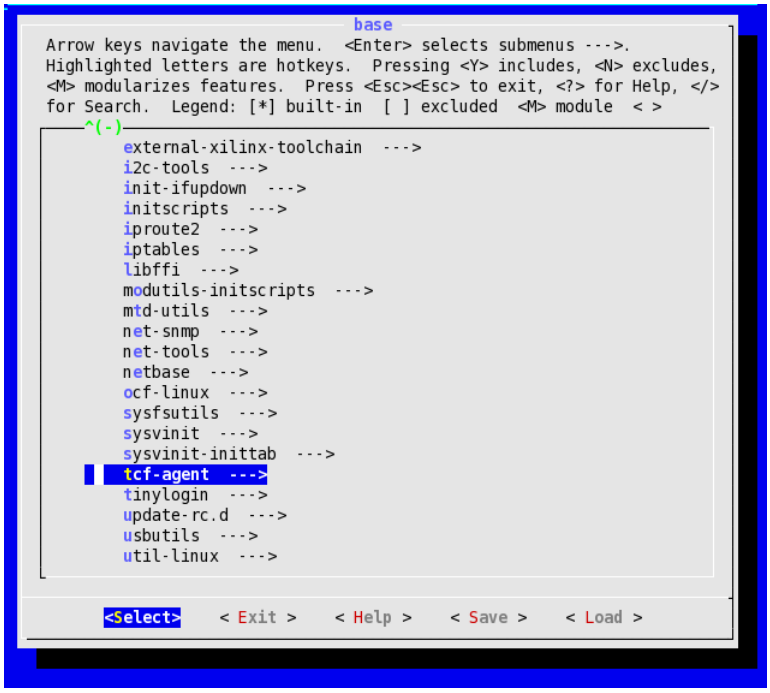

Figure 10: Base Packages sub-menu

5. Exit the menu and select <Yes> to save the configuration.
6. Rebuild the target system image as described previously.

Performing a Debug Session

1. Boot your board (or QEMU) with the new image.
2. Ensure that the boot log indicates that tcf-agent has started. You should see a message like the following;

```
Starting tcf-agent: OK
```

3. Launch XSDK, and create a workspace.
4. Add a Hardware Platform Specification by selecting **File** > **New** > **Project**.
5. In the pop-up window select **Xilinx** > **Hardware Platform Specification**
6. Give the Hardware Project a name. E.g. ZC702
7. Locate the system.xml for your target hardware. If the PetaLinux project was based on a PetaLinux bsp, this can be found in "<project-root>/hardware/Xilinx-ZC702-2013.3/SDK/SDK_Export/hw/system.xml"
8. Open the Debug Launch Configuration window by selecting **Run** > **Debug Configurations**.

9. Create a new `Xilinx C/C++ application (System Debugger)` launch configuration:

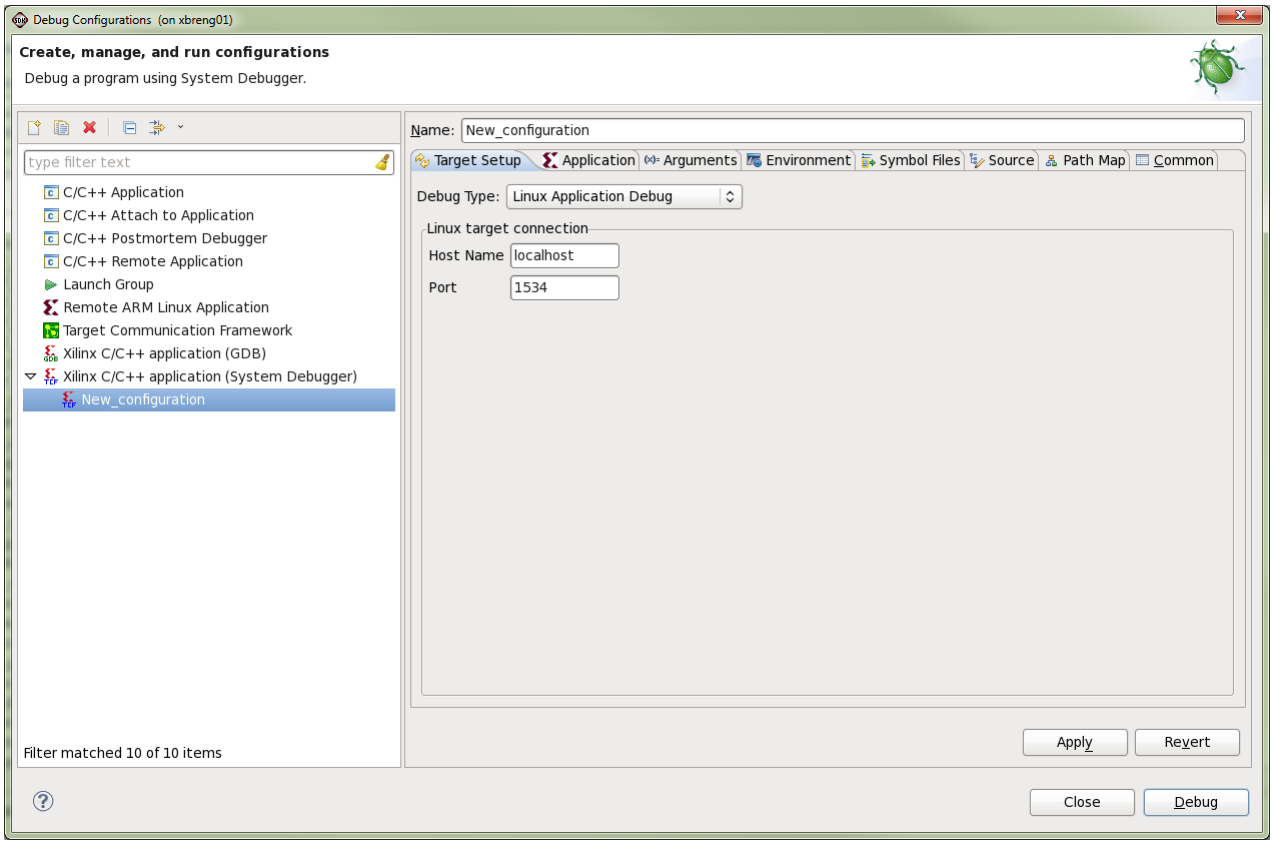

Figure 11: XSDK Debug Configurations

10. Ensure that `Debug Type` is set to `Linux Application Debug`
11. Set the Host Name to the correct IP for the target
12. Set the port of tcf-agent.

IMPORTANT: *If debugging on QEMU, refer to the PetaLinux SDK QEMU System Simulation Guide (UG982) for information regarding IP and port redirection when testing in `non-root` (default) or `root` mode. E.g. if testing in non-root mode, you will need to use `localhost` as the target IP in the subsequent steps.*

13. Switch to the `Application Tab`, and
14. Enter the `Local File Path` to your compiled application in the project directory,
    e.g `"<project-root>/build/linux/rootfs/apps/myapp/myapp"`
15. Enter the `Remote File Path` to the location on the target file system where your application can be found, e.g `"/bin/myapp"`

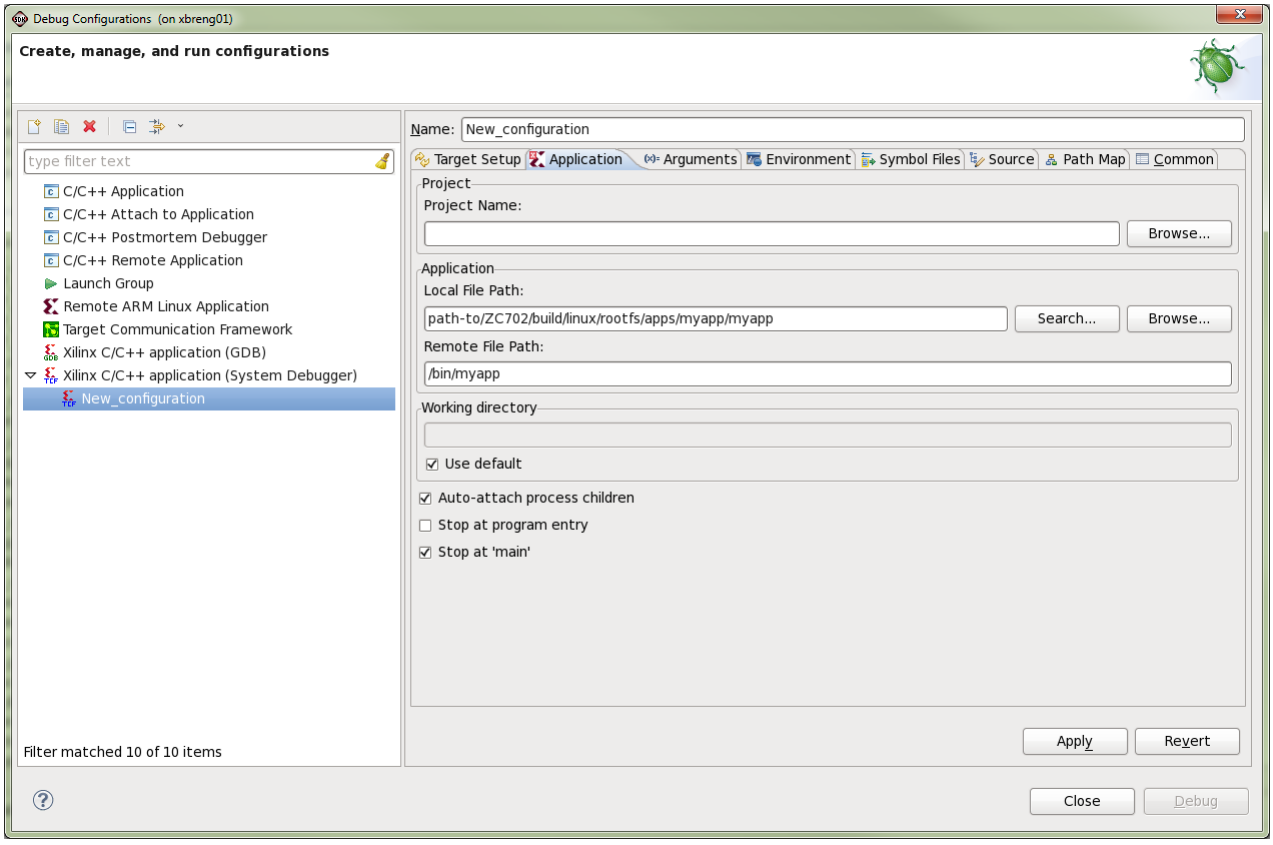

Figure 12: XSDK Debug Configurations

16. Select Debug to Apply the configuration and begin the Debug session. (If asked to switch to Debug Perspective, accept).

17. Standard XSDK debug flow can now commence:

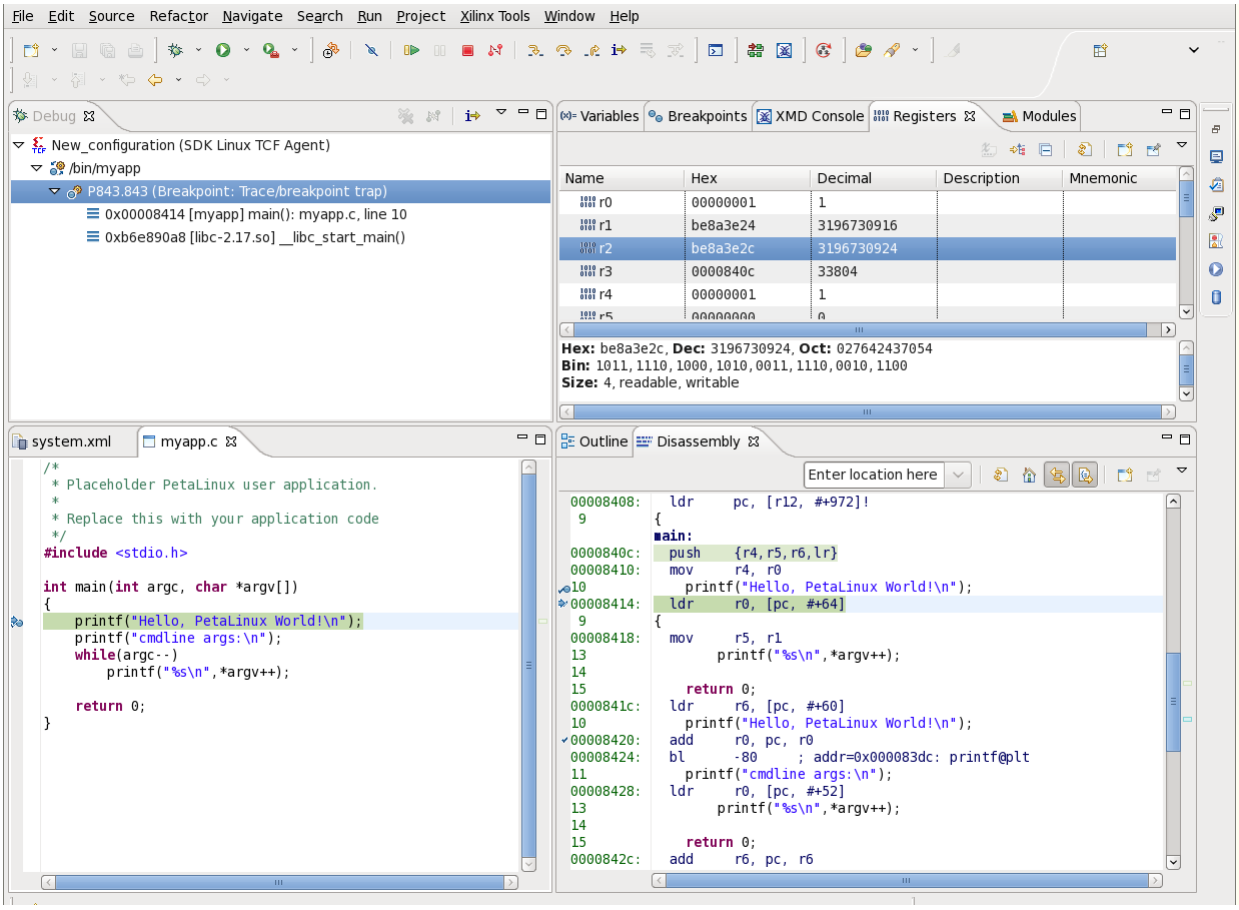

Figure 13: XSDK Debug

Customising the Application Template

In the previous section, you modified the application Makefile for debugging. In this section, we will change the `Kconfig` file and the source file to customise the application template.

As mention in the earlier sections, the `Kconfig` file in the user application directory allows you to configure your user application with Linux `Kconfig` mechanism.

In this section, we will see a simple example of how to change the `Kconfig` template and the source file.

1. Change the source file to print a configurable welcome string. Here is the change (highlighted) to the source file:

```
#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;
}
```

2. Change the `Kconfig` file of the user application to add a configuration option: `APPS_MYAPP_WELCOME`. Here is the change (highlighted) to the `Kconfig` file:

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

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

endif
```

3. Change the user application `Makefile` to pass the configurable option to the user application executable. Here is an example (highlighted):

```
include apps.common.mk

include $(ROOTFS_CONFIG)

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

4. Re-run the application configuration menu:

```
$ petalinux-config -c rootfs
```

5. When the `linux/rootfs Configuration` menu appears, the newly created configuration option will be visible under the following sub menu:

```
Apps --->
    myapp --->
```

You should see the new `Welcome String` configuration on the myapp sub-menu list:

Figure 14: myapp menu with new configuration option

6. Choose the `Welcome String:` option and then input a string, e.g., `"It's a user application test!"`.
7. Exit and save the configuration change.
8. Rebuild application and target system image:

```
$ cd <project-root>
$ petalinux-build -c rootfs/myapp -x clean
$ petalinux-build -c rootfs/myapp
$ petalinux-build -x package
```

9. Boot the new image on hardware or with QEMU. When the system boots, run the new application.

```
root@Xilinx-KC705-AXI-full-2013.3:~# myapp
Hello, It's a user application test!
cmdline args:
myapp
```

One final thing to note, any configuration settings you choose for your application will be saved if you save your default configurations in the top level menuconfig system.

Trouble Shooting

This section describes some common issues you may experience when creating and testing user application, and ways to solve them.

Problem/Error Message	Description and Solution
GDB Error message: `<IP Address>:<port>: Connection refused. GDB cannot connect to the target board using <IP>: <port>`	***Problem Description:*** This error message tells that the GDB client failed to connect to the GDB server. ***Solution:*** 1. Check whether the `gdbserver` is running on the target system. 2. Check whether there is another gdb client already connected to the GDB server. This can be done by looking at the target console. If you can see: `Remote Debugging from host <IP>` it means there is another GDB client connecting to the server. 3. Check whether the IP address and the port is correctly set.

Additional Resources

References

- PetaLinux SDK Application Development Guide (UG981)
- PetaLinux SDK Board Bringup Guide (UG980)
- PetaLinux SDK Firmware Upgrade Guide (UG983)
- PetaLinux SDK Getting Started Guide (UG977)
- PetaLinux SDK Installation Guide (UG976)
- PetaLinux SDK QEMU System Simulation Guide (UG982)

PetaLinux SDK Documentation is available at http://www.xilinx.com/petalinux.