PetaLinux SDK User Guide

Application Development Guide

UG981 (v2013.04) April 22, 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
2013-04-22	2013.04	Updated for PetaLinux SDK 2013.04 release

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.
- At least one PetaLinux BSP (Board Support Package) has been installed.
- You know how to build PetaLinux software image.
- You know how to boot PetaLinux software image.
- 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 shows 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 PetaLinux systems, and install them into the root file system.

The basic steps are as follows:

1. Create a user application by running `petalinux-new-app` on your workstation:

   ```
   $ petalinux-new-app [-t TYPE] <user-application-name>
   ```

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

   ```
   $ petalinux-new-app myapp
   ```

   or:

   ```
   $ petalinux-new-app -t c myapp
   ```

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

   ```
   $ petalinux-new-app -t c++ myapp
   ```

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

   ```
   $ petalinux-new-app -t autoconf myapp
   ```

   You can use `-h` or `--help` to see the usage of the petalinux-new-app. The new application you have created can be found in the `"$PETALINUX/software/user-apps"` directory.

2. Change to the newly created application directory

   ```
   $ cd $PETALINUX/software/user-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 PetaLinux build process. This is not enabled by default. Launch the PetaLinux application configuration menu:

```
$ petalinux-config-apps
```

   `PetaLinux Configuration` menu will show up:

Figure 1: PetaLinux 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 `Custom User Applications`.
3. Press `Enter` to go into the `Custom User Applications` sub-menu:

Figure 2: Custom User Applications Menu

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

4. Select the `myapp` sub-menu, the `myapp` sub-menu will then show up:

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 kernel configuration?`:

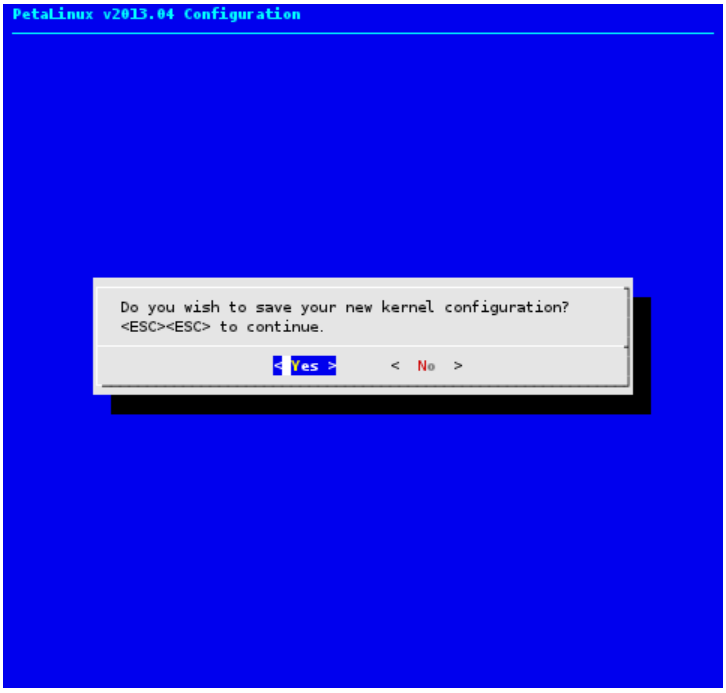

Figure 4: Save New configuration Dialog

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

6. Running make in the "`$PETALINUX/software/petalinux-dist`" directory will rebuild the PetaLinux system image including the selected user application - `myapp`:

```
$ make
```

IMPORTANT:

- *The above user applications build commands should be run in "`$PETALINUX/software/petalinux-dist`" directory.*
- *You should run `make image` after installing the user applications (`make xxxx_romfs`) to update the PetaLinux system image.*

To build all the selected user applications into an complete PetaLinux system image:

```
$ cd $PETALINUX/software/petalinux-dist
$ make userapps_only
$ make userapps_romfs
$ make image
```

To build myapp into an existing PetaLinux system image:

```
$ cd $PETALINUX/software/petalinux-dist
$ make userapps/myapp_only
$ make userapps/myapp_romfs
$ make image
```

TIP: *There are other ways to compile and install the user applications:*

- *to compile all the selected user applications:*

```
$ make userapps_only
```

- *to install all the selected user applications:*

```
$ make userapps_romfs
```

- *to clean all the selected user applications:*

```
$ make userapps_clean
```

- *to compile a specific user application:*

```
$ make userapps/<user-application>_only
```

- *to install a specific user application:*

```
$ make userapps/<user-application>_romfs
```

- *to clean a specific user application:*

```
$ make userapps/<user-application>_clean
```

Testing New Application

So far, you have created and compiled the 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 it, 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 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 PetaLinux system console:

   ```
   # <user-application-executable-file-name>
   ```

   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 PetaLinux filesystem,double check to make sure that you completed the `_romfs` and `image` Make targets described in the previous section. These ensure that your application binary is copied into the root filesystem staging area, and that the PetaLinux system image are updated with this new filesystem.

Debugging Applications with GDB

PetaLinux supports GDB to debug your user application. 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:

   ```
   ...
   ROMFSINST=$(ROOTDIR)/tools/romfs-inst.sh

   CFLAGS += -O0

   APP = myapp
   ...
   ```

2. Change to the "`$PETALINUX/software/petalinux-dist`" directory:

   ```
   $ cd $PETALINUX/software/petalinux-dist
   ```

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

   ```
   $ petalinux-config-apps
   ```

4. Scroll down the PetaLinux Configuration menu to **Debugging**:

Figure 5: PetaLinux Configuration - Scrolling down to Debugging option

5. Select the **Debugging** sub-menu:

Figure 6: Debugging Menu

6. Ensure that `build debuggable applications` and `build debuggable libraries` are selected.

```
[*] build debuggable libraries
[*] build debuggable applications
```

7. Enable gdbserver from the **Root Filesystem Packages** > **base** > **external-sourcery-toolchain** submenu:

```
...
[ ] external-sourcery-toolchain-mtrace
[*] gdbserver
[ ] ldd
...
```

8. Exit the menu and select <Yes> to save the configuration.

9. Rebuild the PetaLinux image by running these commands from the `"$PETALINUX/software/petalinux-dist directory"` (substitute your application name as appropriate)

```
$ make userapps/myapp_clean
$ make userapps/myapp_only
$ make userapps/myapp_romfs
$ make image
```

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 PetaLinux console (defaults listenning on port 1234):

```
# gdbserver host:1234 /bin/myapp
Process /bin/myapp created; pid = 73
Listening on port 1234
```

   1234 is the `gdbserver` port - it can be any unused port number

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

```
$ cd $PETALINUX/software/user-apps/myapp
```

4. Run GDB client

```
$ petalinux-gdb myapp
```

5. The GDB console will start:

```
...
PetaLinux SDK gdb library path auto-configuration
...
(gdb)
```

6. In the GDB console connect to the target machine using the command:

- Use the IP address of the PetaLinux 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 `1234`. If you chose a different gdbserver port number in the earlier step, use that value instead.

```
(gdb) target remote 192.168.0.10:1234
```

The GDB console will attach to the remote target. Here are the messages shown on the target console after attaching to the target.

e.g., the IP address of the workstation is `192.168.0.9`:

```
# gdbserver host:1234 /bin/myapp
Process /bin/myapp created; pid = 73
Listening on port 1234
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 0x10000418: file myapp.c, line 9.
```

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=0xbff35ec4) at myapp.c:9
9 {
```

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

```
(gdb) list
4  * Replace this with your application code
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++);
```

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:

```
~ # gdbserver host:1234 /bin/myapp
Process /bin/myapp created; pid = 58
Listening on port 1234
Remote debugging from host 192.168.0.9
Hello, PetaLinux World!
cmdline args:
/bin/myapp

Child exited with status 0
GDBserver exiting
~ #
```

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.*

GUI Debugging

GDB debugging is supported in the PetaLinux Eclipse SDK, for more details please refer to the *PetaLinux SDK Eclipse Plugin Guide (UG979)* .

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

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: `USER_APPS_MYAPP_WELCOME`. Here is the change (highlighted) to the `Kconfig` file:

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

 config USER_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):

```
ROMFSDIR=$(ROOTDIR)/romfs
ROMFSINST=$(ROOTDIR)/tools/romfs-inst.sh

include $(ROOTDIR)/config/.config
ifneq ($(CONFIG_USER_APPS_MYAPP_WELCOME),)
CFLAGS += -DWELCOME=\"$(CONFIG_USER_APPS_MYAPP_WELCOME)\"
endif
APP = myapp
```

4. Re-run the application configuration menu:

```
$ petalinux-config-apps
```

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

```
Custom User Applications --->
    myapp --->
```

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

Figure 7: 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, filesystem and PetaLinux image:

```
$ make userapps/myapp_only
$ make userapps/myapp_romfs
$ make image
```

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

```
# 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 Eclipse Plugin Guide (UG979)
- PetaLinux SDK Firmware Upgrade Guide (UG983)
- PetaLinux SDK Getting Started Guide (UG977)
- PetaLinux SDK Installation Guide (UG976)
- PetaLinux SDK QEMU System Simulation Guide (UG982)