PetaLinux SDK User Guide

Zynq All Programmable SoC Linux-FreeRTOS AMP Guide

UG978 (v2013.10) November 25, 2013

XILINX®

Revision History

Date	Version	Notes
2012-12-17	2012.12	Initial public release for PetaLinux SDK 2012.12
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

This document details the Linux-FreeRTOS AMP system with PetaLinux and Xilinx Vivado or EDK for Zynq. It includes the following topics:

- Overview of the AMP Reference Design
- Installation of AMP Reference Design
- Getting started with the Reference Design; including the pre-built reference BSP
- How to recreate Linux-FreeRTOS AMP system with PetaLinux

Please note: the reader of this document is assumed to have Linux knowledge such as how to run Linux commands as well as strong familiarity with the PetaLinux tools.

Prerequisites

This document assumes that the following prerequisites have been satisfied:

- PetaLinux SDK has been installed.
- You know how to build a PetaLinux system image.
- You know how to boot a PetaLinux system 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 show the path to the installed PetaLinux. If it shows nothing, please refer to section Environment Setup in the *PetaLinux SDK Getting Started Guide (UG977)* document to setup the environment.

Overview

This section describes the Linux-FreeRTOS AMP reference design system, the components and their configuration.

The Linux-FreeRTOS AMP system is designed to demonstrate Linux's ability to configure the secondary CPU for FreeRTOS and the loading of FreeRTOS firmware. This includes the following components: `remoteproc` drivers, generic `rpmsg` drivers, application specific `rpmsg` drivers and the Trace Buffer.

The `remoteproc` drivers are Linux drivers which control the process of loading and unloading AMP modules on the secondary CPU. This controls the detachment of the secondary CPU from Linux, the associated configuration of the CPU and the loading of the FreeRTOS firmware into the target CPU's memory region.

The `rpmsg` drivers are Linux drivers as well as FreeRTOS library code that controls and manages memory and interrupts for interprocessor communication. transfer data between Linux and FreeRTOS. Figure 1 shows the allocation of these VRING Buffers. The buffers contain messages which are arranged in a specific structure. When messages are placed in a VRING ready for delivery, the sender wil notify the recipient CPU via a software interrupt (also called a 'kick'), routed to the target CPU.

The Trace Buffer is a pre-allocated segment of memory used by FreeRTOS firmware as a log buffer. It allows FreeRTOS to display log messages which can be accessed from Linux without the need for an additional hardware serial console.

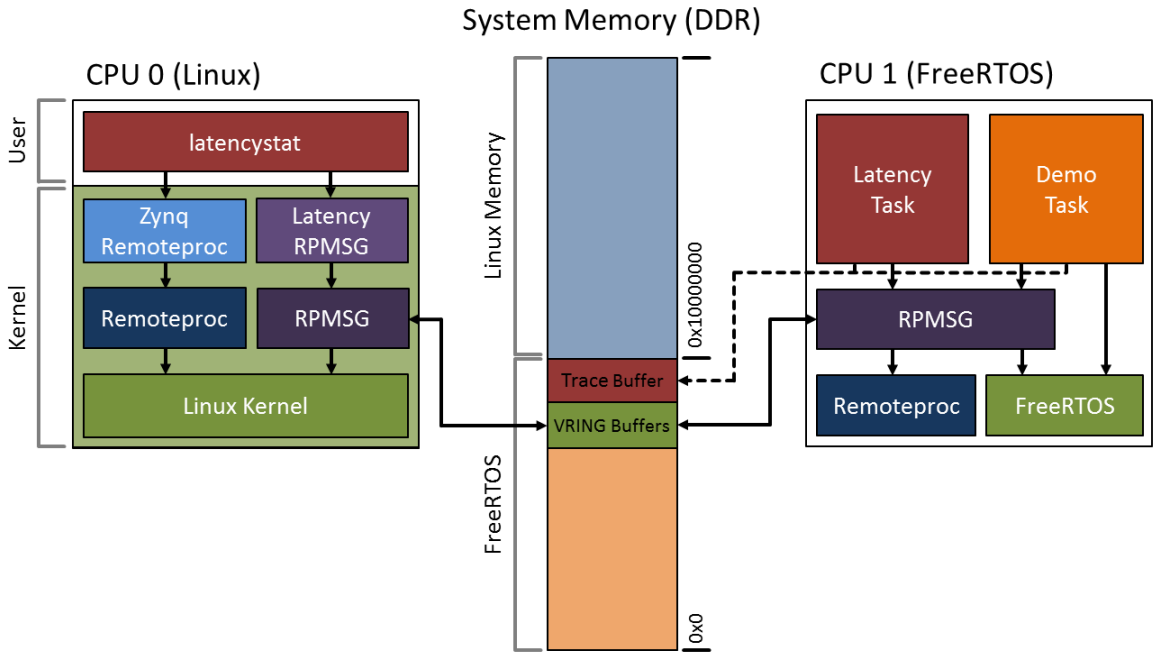

Figure 1: Linux-FreeRTOS AMP Reference Design

FreeRTOS Demo Application

The demo application provided in the reference design demonstrates the use of the `rpmsg` drivers/library for communication of FreeRTOS interrupt latency statistics.

The demonstration FreeRTOS firmware implements two FreeRTOS tasks (as shown in Figure 1).

The first task samples the interrupt latency by configuring the Triple Timer Counter to generate an interrupt on the overflow condition. Once the overflow is hit the timer continues counting. When the interrupt is processed by the FreeRTOS firmware it immediately pauses the timer and reads the current value; this is the approximate time between when the interrupt occurred and when the interrupt was processed, and forms the latency data provided by the demo application.

The second task is a demonstration task which tests its own scheduling to ensure that it meets the expected scheduling jitter. It will print a message to the log buffer as to whether the jitter is as expected or not.

Linux Demo Application

The demo application provided in the reference design called `latencystat` demonstrates the communication between the FreeRTOS application and Linux. This application uses the `rpmsg` drivers to send requests for latency data from the FreeRTOS application and displays this data as output.

Installation

PetaLinux SDK includes the FreeRTOS Repository as well as a Linux Demo Application. These are installed as part of PetaLinux, you can access them from "<petalinux-path>/components/hardware/edk_user_repository/FreeRTOS" and "<petalinux-path>/components/apps/latencystat".

Install the ZC702 (or ZC706) AMP BSP with petalinux-create within your working directory:

```
$ petalinux-create -t project --source <path-to-bsp>/Xilinx-ZC702-AMP-v2013.10-final.bsp
```

After you have executed the create command, you will find multiple reference projects for the BSP. A Vivado 2013.3 based design as well as a EDK/ISE 14.7 based design. In each design will be pre-configured PetaLinux projects as well as pre-built images.

The structure of the projects are as follows:

- The project root e.g. Xilinx-ZC702-AMP-14.7
  - "hardware"
    - Hardware project files generated with Xilinx Vivado or EDK/ISE.
  - "pre-built/linux"
    - "images"
      - "BOOT.BIN" - BIN file composed of FPGA bitstream, FSBL boot loader and u-boot
      - "u-boot.elf" - U-Boot ELF file
      - "image.ub" - Linux kernel in uImage format
      - "zynq_fsbl.elf" - FSBL ELF file
      - "freertos" - FreeRTOS firmware ELF file
    - "implementation"
      - "download.bit" - FPGA bitstream
  - "components", "hw-description", "subsystems", "config.project"
    - PetaLinux project configuration files.

Testing Pre-Built Reference Design

You can test the pre-built images in either QEMU or on hardware. Instructions are provided below.

Boot PetaLinux (Hardware)

For more information on booting a PetaLinux image on a ZC702 or ZC706 Zynq board refer to the *PetaLinux SDK Getting Started Guide (UG977)* section Test Pre-Built PetaLinux Image on Hardware

Boot PetaLinux (QEMU)

For more information on booting a PetaLinux image within a QEMU instance refer to the *PetaLinux SDK Getting Started Guide (UG977)* section Test Pre-Built PetaLinux Image with QEMU

Starting FreeRTOS Firmware

1. Once the system has booted, at the console log in with the username: `root` and password: `root`.
2. Since the second processor hasn't been released by Linux for FreeRTOS yet, the system is still in a conventional SMP state. You can see the 2nd processor from `"/proc/cpuinfo"`:

```
# cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 0 (v7l)
BogoMIPS        : 1332.01
Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xc09
CPU revision    : 0

processor       : 1
model name      : ARMv7 Processor rev 0 (v7l)
BogoMIPS        : 1332.01
Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xc09
CPU revision    : 0

Hardware        : Xilinx Zynq Platform
Revision        : 0000
Serial          : 0000000000000000
```

3. Load the 2nd processor with FreeRTOS firmware as follows by loading the main `zynq_remoteproc` module as well as the firmware module `rpmsg_freertos_statistic`:

```
# modprobe zynq_remoteproc
CPU1: shutdown
 remoteproc0: 0.remoteproc-test is available
 remoteproc0: Note: remoteproc is still under development and considered experimental.
 remoteproc0: THE BINARY FORMAT IS NOT YET FINALIZED, and backward compatibility isn't yet guaranteed.
 remoteproc0: registered virtio0 (type 7)

# modprobe rpmsg_freertos_statistic
 remoteproc0: powering up 0.remoteproc-test
 remoteproc0: Booting fw image freertos, size 2130820
 remoteproc0: remote processor 0.remoteproc-test is now up
virtio_rpmsg_bus virtio0: rpmsg host is online
virtio_rpmsg_bus virtio0: creating channel rpmsg-timer-statistic addr 0x50
rpmsg_freertos_statistic rpmsg0: new channel: 0x400 -> 0x50!
```

4. The 2nd processor is unloaded from Linux and is setup to execute the FreeRTOS firmware.

Demo Application

1. The FreeRTOS firmware samples interrupt latency, the sampled data and control of the firmware is controlled via the `rpmsg` interface. The `latencystat` application provides a example implementation which controls the firmware and transfers the results to Linux which are then displayed in the console.
2. Run `latencystat` demo application as follows:

```
# latencystat -b
```

3. The application will print output similar to the following:

```
Linux-FreeRTOS AMP Demo.
   0: Command 0 ACKed
   1: Command 1 ACKed
Waiting for samples...
   2: Command 2 ACKed
   3: Command 3 ACKed
   4: Command 4 ACKed
------------------------------------------------------------
Histogram Bucket Values:
        Bucket 332 ns (37 ticks) had 14814 frequency
        Bucket 440 ns (49 ticks) had 1 frequency
        Bucket 485 ns (54 ticks) had 1 frequency
        Bucket 584 ns (65 ticks) had 1 frequency
        Bucket 656 ns (73 ticks) had 1 frequency
------------------------------------------------------------
Histogram Data:
        min: 332 ns (37 ticks)
        avg: 332 ns (37 ticks)
        max: 656 ns (73 ticks)
        out of range: 0
        total samples: 14818
------------------------------------------------------------
```

The `latencystat` demo application sends requests to FreeRTOS to ask for latency histogram data. The FreeRTOS will reply with the histogram data, and the latency demo application dumps that data.

The `latencystat` demo application can display the information in a graph format or dump the data in hex. Use the `-h` parameter to display the help information of the application.

Accessing the Trace Buffer

The Trace Buffer is a section of shared memory which is only written to by the FreeRTOS application. This Trace Buffer can be used as a logging console to transfer information to Linux. It can act similar to a one way serial console.

The Trace Buffer is a ring buffer, this means that after the Buffer is full it will wrap around and begin writing to the start of the buffer. When accessing the buffer via Linux it will not be read as a stream. The default Trace Buffer is 32 KB in size.

The Trace Buffer can be accessed via `debugfs` as a file.

```
/sys/kernel/debug/remoteproc/remoteproc0/trace0
```

The trace buffer output, viewed by running the `cat` on the buffer, is shown below.

```
# mount -t debugfs
# cat /sys/kernel/debug/remoteproc/remoteproc0/trace0
Setup TLB for 0:ttc
Setup TLB for address f8000000, TLBptr 103e00
Setup TLB for 1:uart
Setup TLB for address e0000000, TLBptr 103800
Setup TLB for 2:scu
Setup TLB for address f8f00000, TLBptr 103e3c
Protect MMU Table at 100000
Clear TLB for address 100000, TLBptr 100004
FreeRTOS main demo application Dec  4 2012 11:45:39
task_latency: starting sampling of irq latency
task_demo: started
task_demo: task resumed as expected
task_demo: task resumed as expected
rpmsg: CLEAR request
rpmsg: START request
task_demo: task resumed as expected
task_latency: sampled 1 full buffers
task_latency: sampled 1 full buffers
task_demo: task resumed as expected
task_latency: sampled 1 full buffers
task_latency: sampled 1 full buffers
task_latency: sampled 1 full buffers
task_demo: task resumed as expected
...
task_latency: sampled 1 full buffers
task_latency: sampled 1 full buffers
task_latency: sampled 1 full buffers
task_demo: task resumed as expected
rpmsg: STOP request
rpmsg: CLONE request
rpmsg: GET request
rpmsg: QUIT request
```

WARNING: *Communications between FreeRTOS and Linux do not occur with real-time guarantees. Hardware operation, and activity in either OS can delay the delivery of data. All systems should be architected on the basis of no guaranteed hard real-time inter-system communication.*

Bringup a Linux-FreeRTOS AMP System on the ZC702

The following section describes the process to create a Linux-FreeRTOS AMP system with PetaLinux and Xilinx EDK or Vivado.

Please note that the following section describes the bringup process specific to a Linux-FreeRTOS AMP system only. Please refer to *PetaLinux SDK Board Bringup Guide (UG980)* for the details on how to create a project with PetaLinux and how to build PetaLinux.

Covered in this section:

- Hardware Setup (for Vivado and EDK)
- XSDK Setup
  - Create FSBL
  - Create PetaLinux BSP
  - Create FreeRTOS Firmware and BSP
- PetaLinux Configuration and Building
  - Project creation and configuration
  - Kernel configuration
  - Device Tree Setup
  - Installing the Firmware into the filesystem
  - Building and creating bootable Images

Hardware Setup (Vivado)

The ZC702 Development Board Template which is provided with Xilinx Vivado can be used as a base configuration, the template meets the minimum PetaLinux requirements.

WARNING: *Linux requires at least one UART and one storage peripheral (e.g. QSPI, SD, etc.).*

The FreeRTOS BSP requires one serial UART to be selected from the Vivado Zynq Re-customize IP window. The UART port on the ZC702 is connected to **UART 1** which is configured for use by Linux. Enable **UART 0** for use by FreeRTOS and set its IO as EMIO.

The FreeRTOS Demo Application provided uses Timer 1. Enable **Timer 1** and set its IO as EMIO.

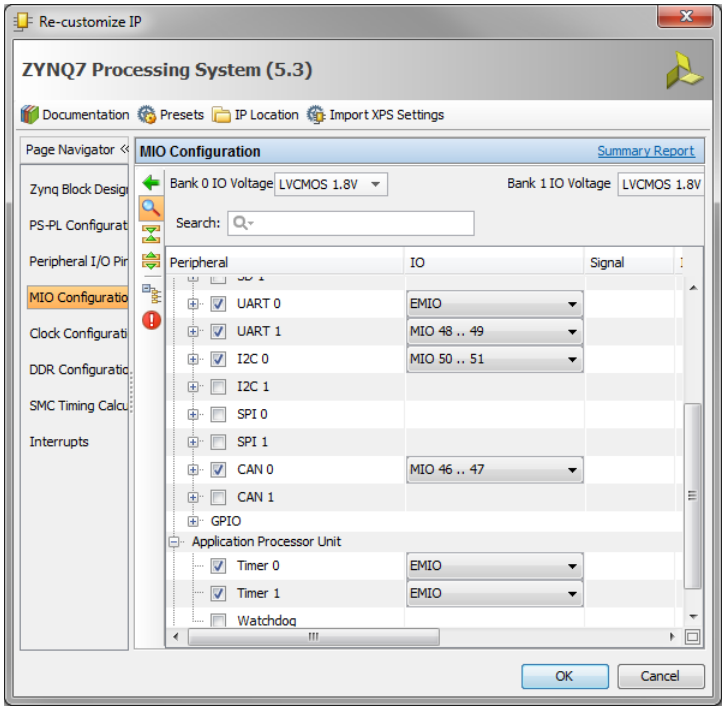

Figure 2: Zynq Processing System Configuration

Once the project is configured, using the **Export Hardware for XSDK** from Vivado to update the hardware description files of an XSDK workspace.

Hardware Setup (EDK)

The ZC702 Development Board Template which is provided with Xilinx EDK can be used as a base configuration, the template meets the minimum PetaLinux requirements.

WARNING: *Linux requires at least one UART and one storage peripheral (e.g. QSPI, SD, etc.).*

The FreeRTOS BSP requires one serial UART to be selected from the XPS Zynq PS MIO Configurations wizard. The UART port on the ZC702 is connected to **UART 1** which is configured for use by Linux. Enable **UART 0** for use by FreeRTOS and set its IO as EMIO.

The FreeRTOS Demo Application provided uses Timer 1. Enable **Timer 1** and set its IO as EMIO.

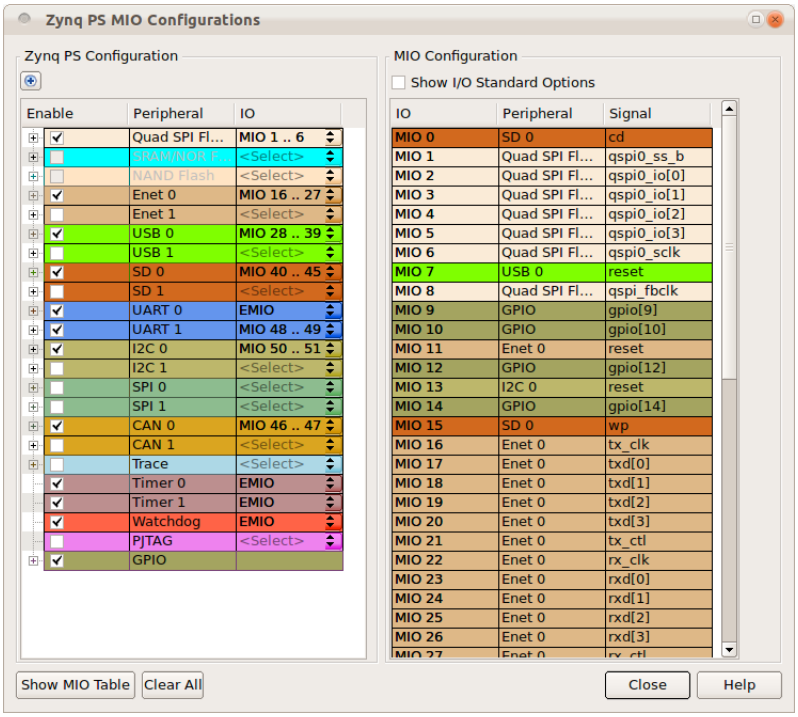

Figure 3: Zynq Peripheral Configuration

Once the project is configured, be sure to select **Export Design to XSDK** from XPS to update the hardware description files of XSDK workspace.

XSDK Setup

XSDK is used to build and prepare the PetaLinux BSP, FSBL and the FreeRTOS BSP and application.

Setup Workspace

When you open your XSDK workspace, you will need to add PetaLinux and FreeRTOS BSP repositories as follows:

1. Go to **Xilinx Tools > Repositories** to add the following repositories:
   - `"<petalinux-path>/components/hardware/edk_user_repository"`
   - `"<petalinux-path>/components/hardware/edk_user_repository/FreeRTOS"`
   - `"<petalinux-path>/components/hardware/edk_user_repository/FreeRTOS/drivers"`
   - `"<petalinux-path>/components/hardware/edk_user_repository/FreeRTOS/bsp"`
2. Click **Rescan Repositories**
3. Click **Apply**
4. Click **Ok**

Create FSBL

Create a Xilinx FSBL application project as follows:

1. Go to **File > New > Application Project**
2. Name the project (e.g. **FSBL**)
3. Select `ps7_cortexa9_0` as **Processor** from the New Project wizard
4. Click **Next**
5. Select **Zynq FSBL** as the **Select Project Template**
6. Click **Finish** to create the project

Create PetaLinux BSP

Create a PetaLinux BSP as follows:

1. Go to **File > New > Board Support Package**
2. Select `ps7_cortexa9_0` as the **CPU** in the **New Board Support Package Project** wizard
3. Select **petalinux** as **Board Support Package OS**

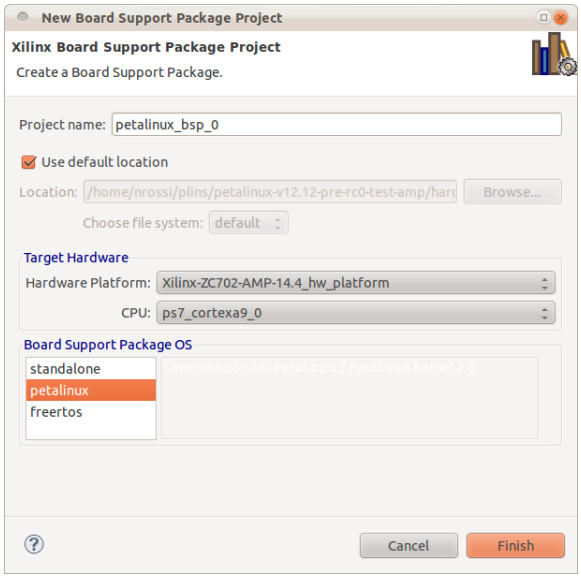

Figure 4: New BSP Wizard - PetaLinux

4. Click **Finish**. A Board Support Package Settings window will pop up.
5. Select `petalinux` from the **Overview** in the **Board Support Package Settings** window
6. Select `ps7_uart_1` as `stdout` and `stdin` in the **Configuration for OS table**
7. Select `ps7_ddr_0` as the `main_memory`

8. Select `ps7_qspi_0` as the `flash_memory`
9. Select `ps7_sd_0` as the `sdio`
10. Select `ps7_ethernet_0` as the `ethernet`

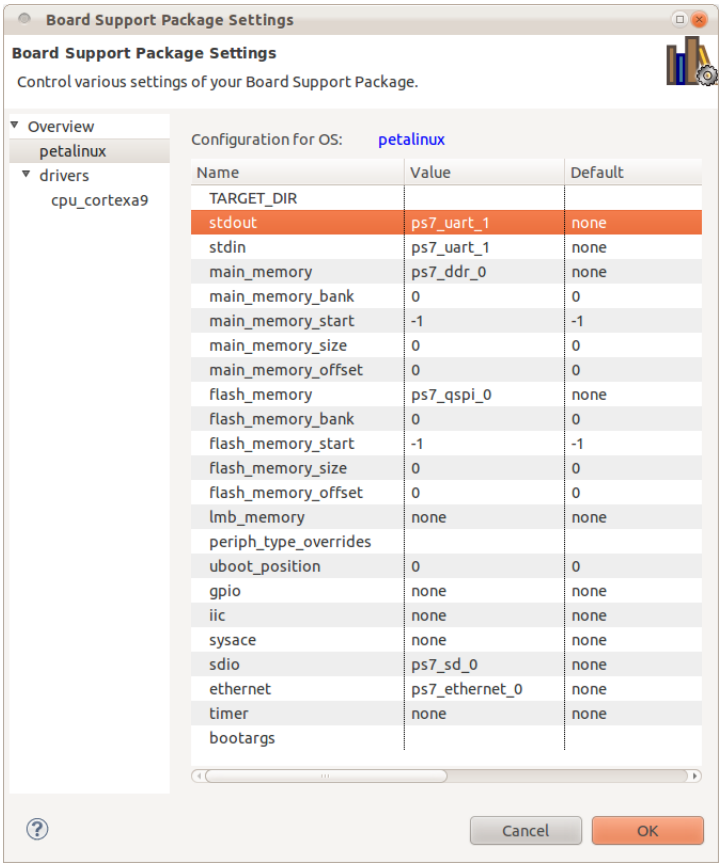

Figure 5: PetaLinux BSP Configuration

11. Click **Ok**

Create FreeRTOS BSP

Create a FreeRTOS BSP as follows:

1. Go to **File > New > Board Support Package**
2. Select `ps7_cortexa9_1` as the **CPU** in the **New Board Support Package Project** wizard
3. Select `freertos` as **Board Support Package OS**

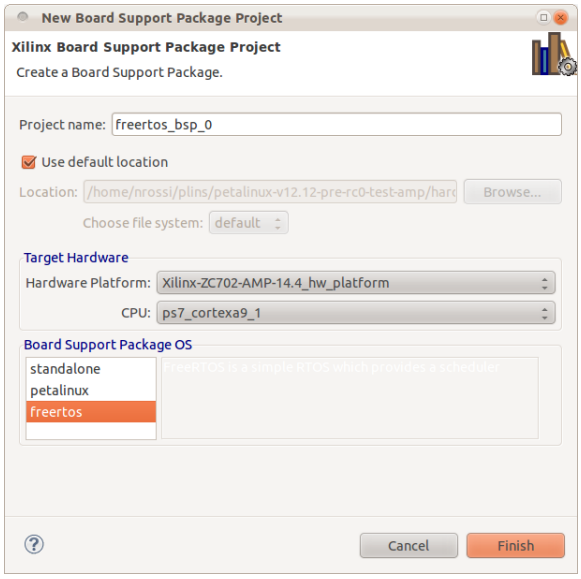

Figure 6: New BSP Wizard - FreeRTOS

4. Click **Finish**. A Board Support Package Settings window will pop up.
5. Select `freertos` from the **Overview** in the **Board Support Package Settings** window
6. Select `ps7_uart_0` as `stdout` and `stdin` in the **Configuration for OS table**

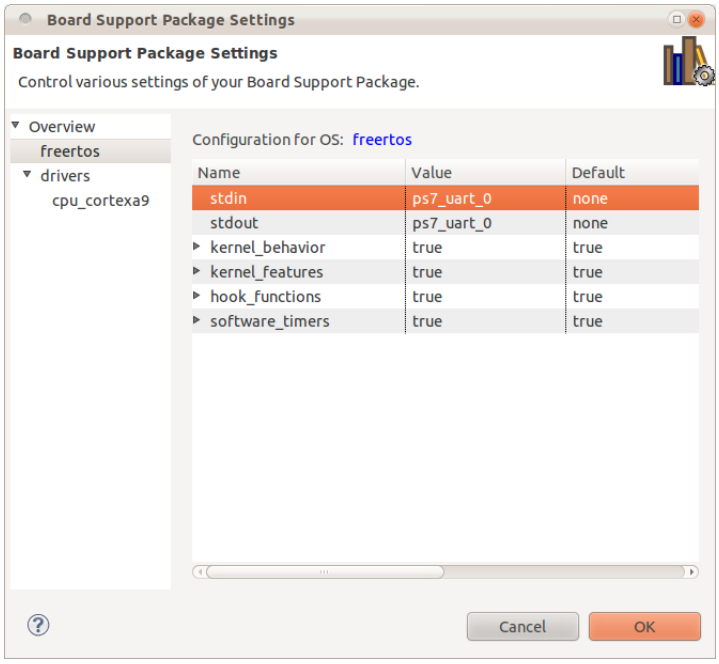

Figure 7: FreeRTOS BSP Configuration

7. Select `cpu_cortexa9` from **Overview > drivers**

8. Add `-DUSE_AMP=1` to the `extra_compiler_flags` of the **Configuration for driver** table. This will enable AMP specific features in the FreeRTOS firmware.

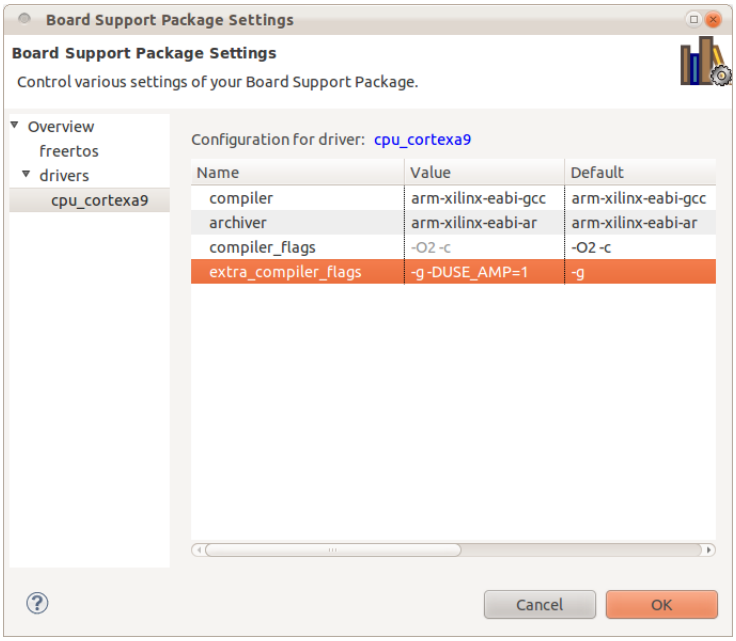

Figure 8: FreeRTOS BSP Configuration

9. Click **Ok**

Create FreeRTOS Application

Create a FreeRTOS AMP application project as follows:

- Go to **File > New > Application Project**
- Name the project (e.g. **freertos_amp_demo_application**)
- Select `ps_cortexa9_1` as **Processor** from the New Project wizard
- Select **Board Support Package**, **Use Existing** and select the existing BSP created in the previous section.

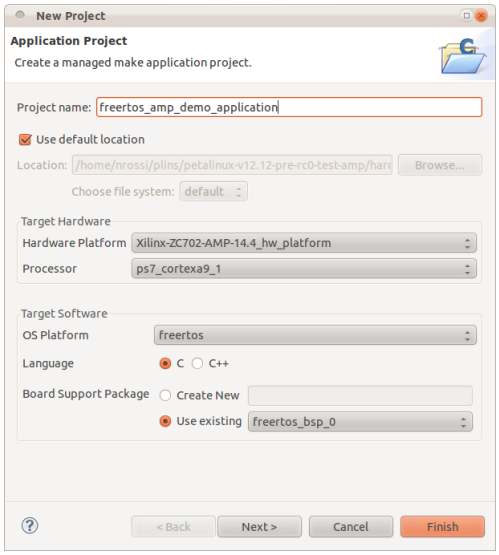

Figure 9: New Application Project - FreeRTOS Demo Application

- Click **Next**
- Select **FreeRTOS AMP** as the template. You are free to modify this FreeRTOS AMP application template for your own purpose.

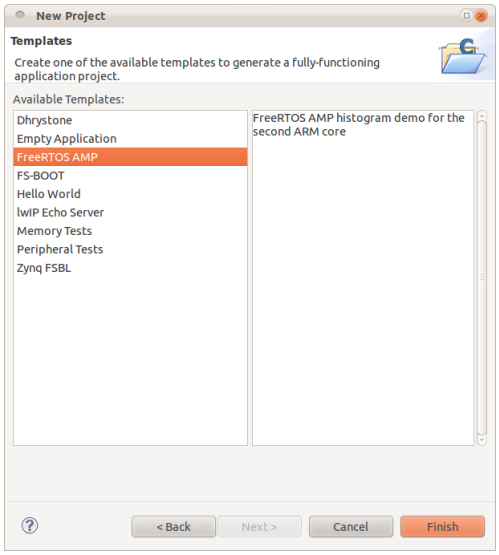

Figure 10: New Application Project - FreeRTOS Demo Application

- Click **Finish**

PetaLinux Configuration

So far, we have generated a FSBL project, a PetaLinux BSP and a FreeRTOS AMP application with XSDK. In this section, we are going to configure PetaLinux software platform to support AMP.

Project Setup

At first, we will create a software platform with the PetaLinux tool. Specify a project name when creating the project (e.g. AMP-Demo).

```
$ petalinux-create -t project -n AMP-Demo
```

The `petalinux-create` command will create a new project (the default architecture is Zynq).

Project Configuration

Use `petalinux-config` to import your hardware settings into the project (see the *PetaLinux SDK Board Bringup Guide (UG980)* for further details).

```
$ petalinux-config --get-hw-description -p <project-root>
```

Memory Space Configuration

For the AMP system memory is shared between the Linux Kernel and the FreeRTOS application, PetaLinux must be configured to segment the memory. In this example the first 256MB of memory is dedicated to the FreeRTOS application and starts at address `0x0` up to address `0xfffffff`.

This is achieved by configuring the kernel to start at address `0x10000000`. The kernel base address is configured as part of the main project configuration.

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

Change the **Kernel base address** value to be `0x10000000`

```
    *** linux Components Selection ***
...
(0x10000000) Kernel base address
```

Kernel Configuration

1. Change into the directory of your project.

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

2. Configure the kernel using:

   ```
   $ petalinux-config -c kernel
   ```

3. Select **Enable loadable module support**.

   ```
   Kernel Configuration --->
       [*] Enable loadable module support  --->
   ```

4. Select **High Memory Support** within **Kernel Features**
5. Select **2G/2G user/kernel split** as **Memory split** within **Kernel Features**.

```
Kernel Configuration --->
    Kernel Features --->
            Memory split (2G/2G user/kernel split)  --->
        [*] High Memory Support
```

6. Enable **Userspace firmware loading support**:

```
Kernel Configuration --->
    Device Drivers --->
        Generic Driver Options --->
            <*> Userspace firmware loading support
            [ ]   Include in-kernel firmware blobs in kernel binary
            () External firmware blobs to build into the kernel binary
```

7. Enable **Remoteproc** and **Rpmsg** drivers:

```
Kernel Configuration --->
    Device Drivers --->
        Remoteproc drivers (EXPERIMENTAL) --->
            <M> Support ZYNQ remote proc
        Rpmsg drivers (EXPERIMENTAL) --->
            <M> rpmsg OMX driver
            <M> An FreeRTOS statistic
```

   Enable the the drivers as a Module **<M>**.

8. Exit the menuconfig and save changes.

Root File-system Configuration

1. Change into the directory of your project.

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

2. Configure the File-system to include the demo applications:

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

3. Select the Linux AMP demo application and the FreeRTOS demo application. The **latencystat** demo can be found in the **Apps** submenu.

```
Apps --->
    [*] latencystat  --->
```

DTS (Device Tree Source) Setup

The Linux Kernel is driven by device trees. The `remoteproc` driver is also instantiated and configured by a device tree node. The DTS file inside the PetaLinux project is generated from a hardware description and requires modification to support AMP.

Open the DTS file (with a text editor) located in the project directory.

```
<project-root>/subsystems/linux/hw-description/system.dts
```

Add a device node for the Zynq `remoteproc` driver so that the driver can be probed. You can add the device node to the end of the DTS file:

```
test: remoteproc-test@0 {
        compatible = "xlnx,zynq_remoteproc";
        reg = < 0x0 0x10000000 >;
        interrupt-parent = <&ps7_scugic_0>;
        interrupts = < 0 37 4 0 38 4 >;
        firmware = "freertos";
        ipino = <6>;
        vring0 = <2>;
        vring1 = <3>;
} ;
```

- The `compatible` property must match the one in the driver.
- The `reg` property is the memory segment for the firmware.
- The `interrupts` property allows the consumption of interrupts from Linux to be routed for FreeRTOS, in this case the TTC1 interrupts are routed.
- The `firmware` property is the default name of the firmware.
- The `ipino` property is the Inter-Processor Interrupt (IPI) from firmware to Linux,
- `vring0` and `vring1` properties are the IPIs from Linux to firmware.

Build PetaLinux with AMP support

The previous sections cover the compilation and configuration of the FreeRTOS application and PetaLinux. This section will cover the process of building and preparing bootable images.

Install the Firmware Image

The firmware image for the FreeRTOS application must be placed in the firmware directory of the root filesystem `"/lib/firmware/"`. In order to achieve this a user application must be created.

1. Change into the directory of your project.

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

2. Create the user application:

```
$ petalinux-create -t apps --template install -n freertos_fw
```

3. Copy the FreeRTOS application binary into the user applications data directory.

```
$ cd <project-root>/components/apps/freertos_fw
$ cp <FreeRTOS-application-directory>/Debug/<application>.elf data/freertos
```

4. Edit the user application Makefile, so that during the `install` target the binary is copied into the correct location.

```
include $(PETALINUX)/components/apps/apps.common.mk

FIRMWARE=freertos
install:
        $(TARGETINST) -d data/$(FIRMWARE) /lib/firmware/${FIRMWARE}
```

5. Enable the custom user application:

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

   Update the **Apps** menu select the user application:

```
Apps --->
    [*] freertos_fw  --->
```

Build PetaLinux

Execute the PetaLinux build to compile the kernel, u-boot, root filesystem and user applications.

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

Create the BOOT.BIN

The generated kernel image and u-boot images are in the "<project-root>/images/linux/" directory. In order to create a boot image, use the petalinux-package from within the projects root directory to generate the "BOOT.BIN".

```
$ petalinux-package --boot --fsbl <path-to-FSBL> -f <path-to-FPGA-bitstream>
```

The "BOOT.BIN" file will packaged to the current working directory. Now that all images are created you can boot the system, and follow the runtime instructions from *Testing Pre-Built Reference Design* to load and run the your project.

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.