PetaLinux SDK User Guide

Zynq AMP Linux FreeRTOS Guide

UG978 (v2013.04) April 22, 2013

XILINX®

Revision History

Date	Version	Notes
2012-12-17	2012.12	Initial public release for PetaLinux SDK 2012.12
2013-04-22	2013.04	Updated for PetaLinux SDK 2013.04 release

Table of Contents

About this Guide

This document details the Linux FreeRTOS AMP system with PetaLinux and Xilinx 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 `Getting Started with PetaLinux SDK` 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. This is implemented with pre-allocated memory segments which are used for the transfer of data to and from Linux/FreeRTOS. Figure 1 shows the allocation of these VRING Buffers. The buffers contain messages which are arranged in a specific structure. Once messages are created and stored in the VRING buffer the sender will issue a 'kick' to the recipient CPU, the 'kick' is a software interrupt generated by the Zynq Software Generated Interrupts and is 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.

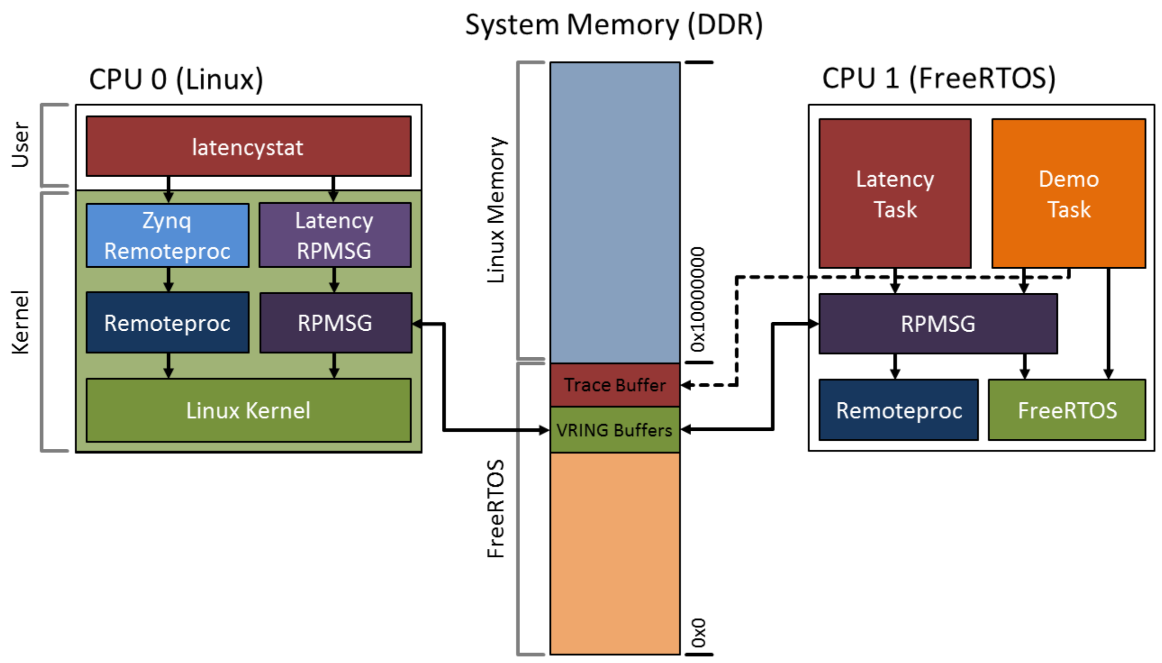


Figure 1: Linux-FreeRTOS AMP Reference Design

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 FreeRTOS firmware is implemented as 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 is provided with the FreeRTOS BSP Repository as well as a Linux Demo Application. These are installed as part of PetaLinux, you can access them from "`$PETALINUX/hardware/edk_user_repository/FreeRTOS`" and "`$PETALINUX/software/demo-apps/latencystat`".

Install the ZC702 AMP reference design with `petalinux-install-bsp` as follows inside PetaLinux tree:

```
$ petalinux-install-bsp <path-to-bsp>/Xilinx-ZC702-AMP-14.5.bsp
```

After you have installed the BSP, you will find the ZC702 AMP reference design from "`$PETALINUX/hardware/reference-designs/Xilinx-ZC702-AMP-14.5/`" directory. Inside this directory, you will also find the hardware project files and the pre-built images.

The structure of the directory is as follows:

- `Xilinx-ZC702-AMP-14.5`
  - Hardware project files generated with Xilinx EDK such as "`system.mhs`", "`system.xmp`", etc.
  - "`pre-built`"
    - "`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
    - "`sw-bsp`"
      - PetaLinux configuration files and DTS (Device Tree Source)

Testing Pre-Built Reference Design

You can test the pre-build images as follows:

Boot PetaLinux

1. Configure the ZC702 to use SD boot mode by connecting 1-2 of jumper J22 and J25 on the board.
2. Connect the UART port on ZC702 to your host
3. Connect the Ethernet port on ZC702 to a local network
4. Copy the "`BOOT.BIN`" and "`image.ub`" from the pre-built images directory: "`Xilinx-ZC702-AMP-14.5/pre-built/images/`" to a SD card.
5. Insert the SD card into the SD card slot on ZC702 and then power on the board.
6. Use a serial terminal application such as Kermit to monitor the UART output from ZC702. Configure the terminal application to use a baudrate of 115200-8N1.

Starting FreeRTOS Firmware

1. Watch the console. Log into PetaLinux with username: `root` and password: `root`.
2. Load the `remoteproc` modules to prepare to load the 2nd processor with FreeRTOS firmware from the PetaLinux console as follows:

```
# modprobe virtio
# modprobe virtio_ring
# modprobe virtio_rpmsg_bus
# modprobe rpmsg_proto
# modprobe remoteproc
```

3. Since the 2nd processor hasn't been unloaded by Linux for use by FreeRTOS yet, the system is still a SMP system. You can see the 2nd processor from "`/proc/cpuinfo`":

```
# cat /proc/cpuinfo
Processor : ARMv7 Processor rev 0 (v7l)
processor : 0
BogoMIPS : 1332.01

processor : 1
BogoMIPS : 1332.01

Features : swp half thumb fastmult vfp edsp neon vfpv3 tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant : 0x3
CPU part : 0xc09
CPU revision : 0

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

4. Load the 2nd processor with FreeRTOS firmware as follows:

```
# modprobe zynq_remoteproc
```

5. You can see the following messages on the console:

```
NET: Registered protocol family 40
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: powering up 0.remoteproc-test
 remoteproc0: Booting fw image freertos, size 2310301
 remoteproc0: remote processor 0.remoteproc-test is now up
virtio_rpmsg_bus virtio0: rpmsg host is online
 remoteproc0: registered virtio0 (type 7)
virtio_rpmsg_bus virtio0: creating channel rpmsg-timer-statistic addr 0x50
```

Demo Application

1. The FreeRTOS application provided in the pre-built reference design collects interrupt latency statistics within the FreeRTOS environment, and reports the results to Linux which are displayed by the `latencystat` Linux demo application. The `rpmsg_freertos_statistic` module must first be loaded so that we can send/receive messages to FreeRTOS. To load the module run the following command in the PetaLinux console:

```
# modprobe rpmsg_freertos_statistic
```

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 following is the output seen in the trace buffer when running a `cat` on the buffer.

```
# 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_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_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: *The Trace Buffer has a known issue where the contents of the buffer is delayed between access in FreeRTOS and Linux. This is a known issue, see the section Trace Buffer is slow for more information.*

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.

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.

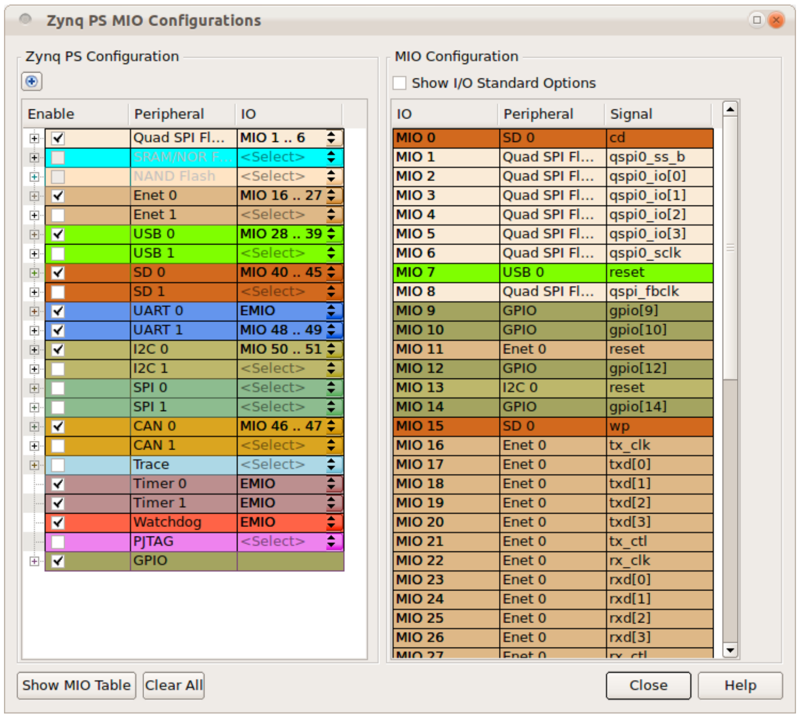
Hardware Setup

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: *PetaLinux 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.



Enable	Peripheral	IO
✓	Quad SPI Fl...	MIO 1 .. 6
	SRAM/NOR F...	<Select>
	NAND Flash	<Select>
✓	Enet 0	MIO 16 .. 27
	Enet 1	<Select>
✓	USB 0	MIO 28 .. 39
	USB 1	<Select>
✓	SD 0	MIO 40 .. 45
	SD 1	<Select>
✓	UART 0	EMIO
✓	UART 1	MIO 48 .. 49
✓	I2C 0	MIO 50 .. 51
	I2C 1	<Select>
	SPI 0	<Select>
	SPI 1	<Select>
✓	CAN 0	MIO 46 .. 47
	CAN 1	<Select>
	Trace	<Select>
✓	Timer 0	EMIO
✓	Timer 1	EMIO
✓	Watchdog	EMIO
	PJTAG	<Select>
✓	GPIO	

IO	Peripheral	Signal
MIO 0	SD 0	cd
MIO 1	Quad SPI Fl...	qspi0_ss_b
MIO 2	Quad SPI Fl...	qspi0_io[0]
MIO 3	Quad SPI Fl...	qspi0_io[1]
MIO 4	Quad SPI Fl...	qspi0_io[2]
MIO 5	Quad SPI Fl...	qspi0_io[3]
MIO 6	Quad SPI Fl...	qspi0_sclk
MIO 7	USB 0	reset
MIO 8	Quad SPI Fl...	qspi_fbclk
MIO 9	GPIO	gpio[9]
MIO 10	GPIO	gpio[10]
MIO 11	Enet 0	reset
MIO 12	GPIO	gpio[12]
MIO 13	I2C 0	reset
MIO 14	GPIO	gpio[14]
MIO 15	SD 0	wp
MIO 16	Enet 0	tx_clk
MIO 17	Enet 0	txd[0]
MIO 18	Enet 0	txd[1]
MIO 19	Enet 0	txd[2]
MIO 20	Enet 0	txd[3]
MIO 21	Enet 0	tx_ctl
MIO 22	Enet 0	rx_clk
MIO 23	Enet 0	rxd[0]
MIO 24	Enet 0	rxd[1]
MIO 25	Enet 0	rxd[2]
MIO 26	Enet 0	rxd[3]
MIO 27	Enet 0	rx_ctl

Figure 2: 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:

WARNING: *Ensure that the XSDK workspace is within the PetaLinux tree, this is required for the PetaLinux tools. (e.g. "`$PETALINUX/hardware/user-platforms/<hw-project-name>/workspace`")*

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

Create FSBL

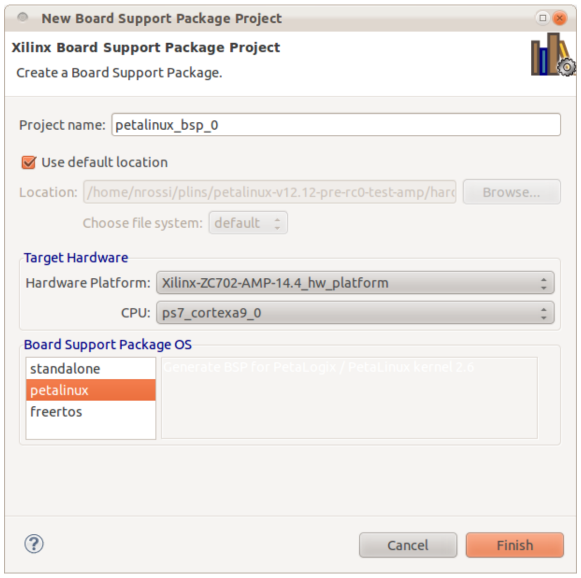
Create a Xilinx FSBL application project as follows:

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**



Xilinx Board Support Package Project

Create a Board Support Package.

Project name: petalinux_bsp_0

[✓] Use default location

Location: /home/nrossi/plins/petalinux-v12.12-pre-rc0-test-amp/harc

Choose file system: default

Target Hardware

Hardware Platform: Xilinx-ZC702-AMP-14.4_hw_platform

CPU: ps7_cortexa9_0

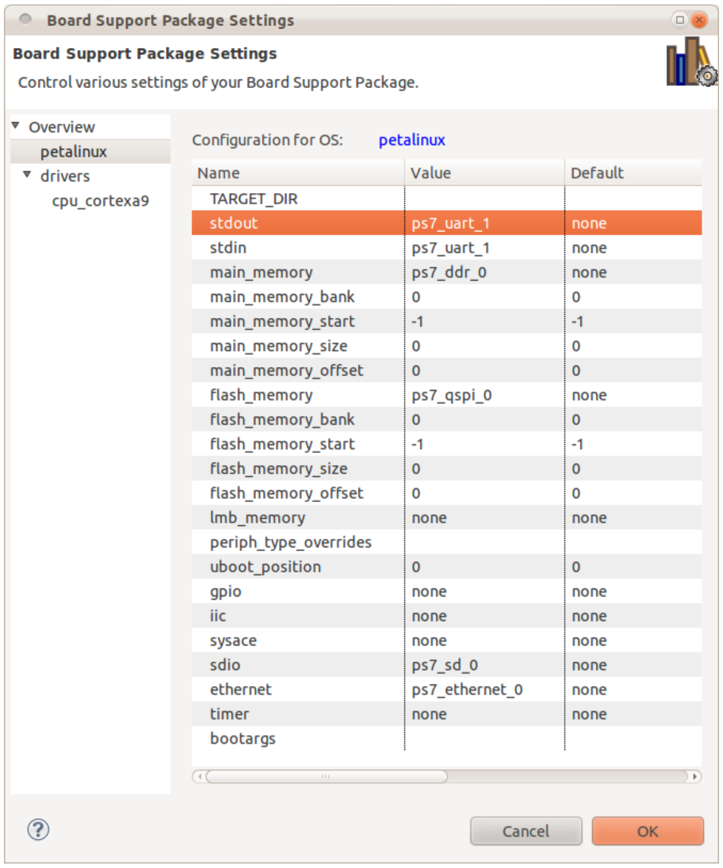
Board Support Package OS

standalone
[✓ petalinux]
freertos

Generate BSP for PetaLogix / PetaLinux kernel 2.6

Figure 3: 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`



Board Support Package Settings

Control various settings of your Board Support Package.

Configuration for OS: petalinux

Name	Value	Default
TARGET_DIR		
stdout	ps7_uart_1	none
stdin	ps7_uart_1	none
main_memory	ps7_ddr_0	none
main_memory_bank	0	0
main_memory_start	-1	-1
main_memory_size	0	0
main_memory_offset	0	0
flash_memory	ps7_qspi_0	none
flash_memory_bank	0	0
flash_memory_start	-1	-1
flash_memory_size	0	0
flash_memory_offset	0	0
lmb_memory	none	none
periph_type_overrides		
uboot_position	0	0
gpio	none	none
iic	none	none
sysace	none	none
sdio	ps7_sd_0	none
ethernet	ps7_ethernet_0	none
timer	none	none
bootargs		

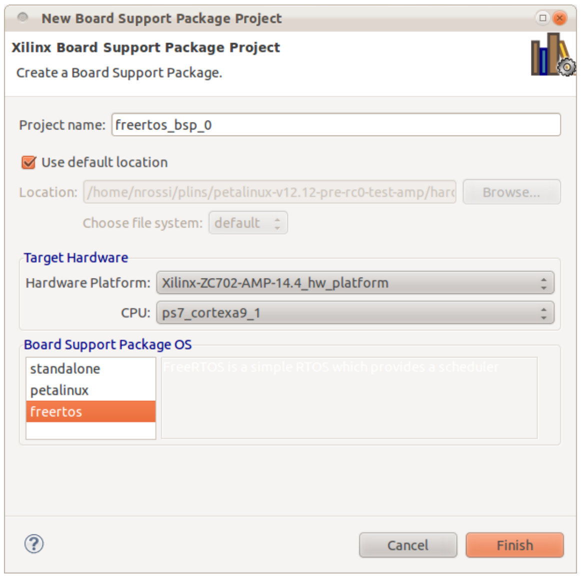
Figure 4: PetaLinux BSP Configuration

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**



New Board Support Package Project

Xilinx Board Support Package Project

Create a Board Support Package.

Project name:	freertos_bsp_0
Use default location	✓
Location:	/home/nrossi/plins/petalinux-v12.12-pre-rc0-test-amp/harc
Choose file system:	default

Target Hardware

Hardware Platform:	Xilinx-ZC702-AMP-14.4_hw_platform
CPU:	ps7_cortexa9_1

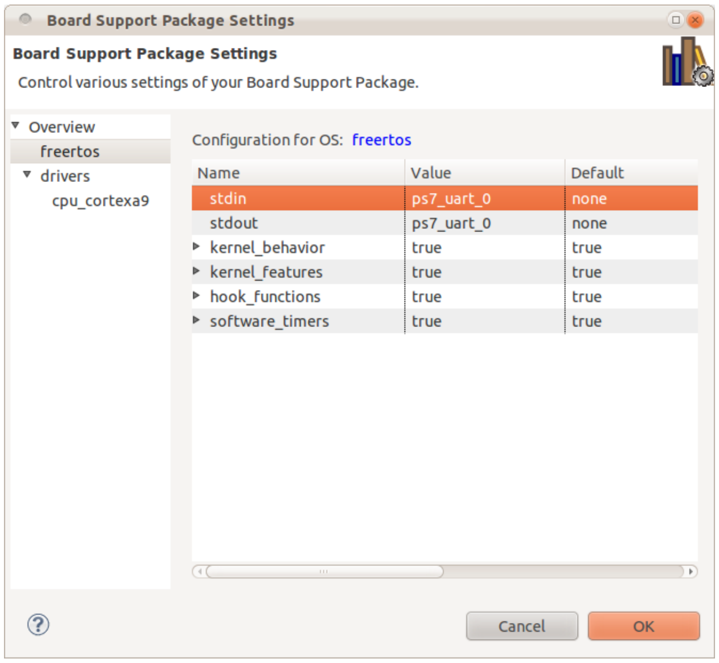
Board Support Package OS

- standalone
- petalinux
- freertos

FreeRTOS is a simple RTOS which provides a scheduler

Figure 5: 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**



Board Support Package Settings

Board Support Package Settings

Control various settings of your Board Support Package.

- Overview
  - freertos
  - drivers
    - cpu_cortexa9

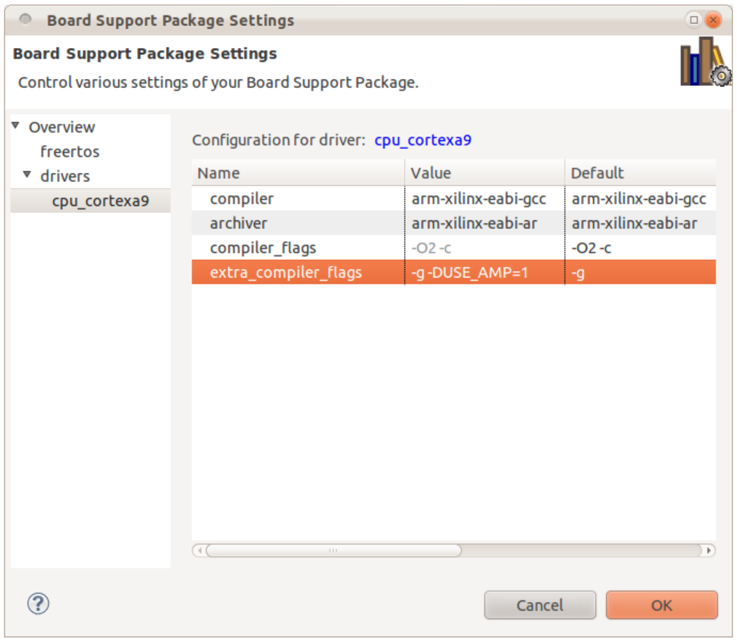
Configuration for OS: freertos

Name	Value	Default
stdin	ps7_uart_0	none
stdout	ps7_uart_0	none
kernel_behavior	true	true
kernel_features	true	true
hook_functions	true	true
software_timers	true	true

Figure 6: 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.



Board Support Package Settings

Control various settings of your Board Support Package.

Configuration for driver: cpu_cortexa9

Name	Value	Default
compiler	arm-xilinx-eabi-gcc	arm-xilinx-eabi-gcc
archiver	arm-xilinx-eabi-ar	arm-xilinx-eabi-ar
compiler_flags	-O2 -c	-O2 -c
extra_compiler_flags	-g -DUSE_AMP=1	-g

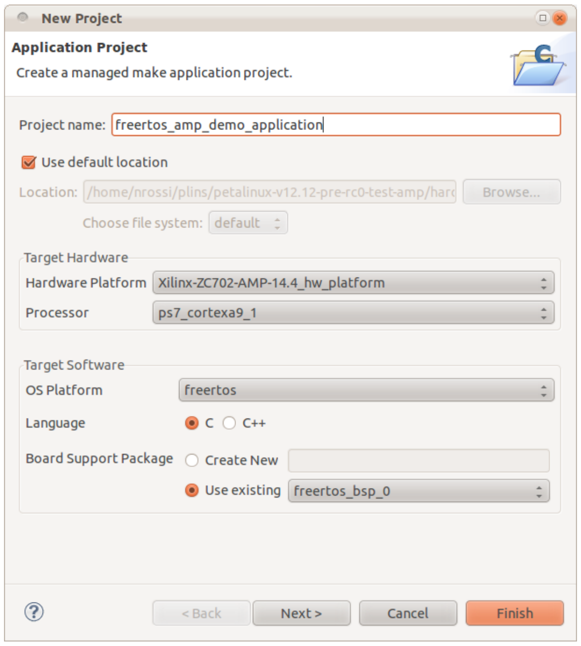
Figure 7: FreeRTOS BSP Configuration

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.


New Project

Application Project

Create a managed make application project.

Project name: freertos_amp_demo_application

[✓] Use default location

Location: /home/nrossi/plins/petalinux-v12.12-pre-rc0-test-amp/harc

Choose file system: default

Target Hardware

Hardware Platform: Xilinx-ZC702-AMP-14.4_hw_platform

Processor: ps7_cortexa9_1

Target Software

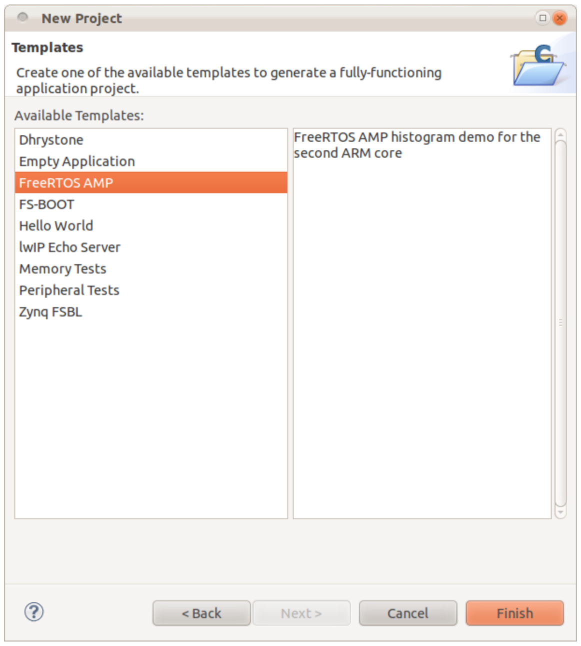
OS Platform: freertos

Language: [✓ C] C++

Board Support Package: Create New [✓ Use existing] freertos_bsp_0

Figure 8: 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.


New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

Dhrystone
Empty Application
FreeRTOS AMP
FS-BOOT
Hello World
lwIP Echo Server
Memory Tests
Peripheral Tests
Zynq FSBL

FreeRTOS AMP histogram demo for the second ARM core

Figure 9: 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.

Platform Setup

At first, we will create a software platform with the PetaLinux tool. Specify a vendor name and platform name to use when creating the platform.

```
$ petalinux-new-platform -c arm -v Demo -p ZC702-AMP-14.5
```

The `petalinux-new-platform` command will create and select the new platform. Use `petalinux-copy-autoconfig` to import your hardware settings in the new platform (see the PetaLinux Board Bringup Guide for further details).

Kernel Configuration

1. Configure the kernel using `petalinux-config-kernel`.

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

2. Set **Physical address of main memory** to `0x10000000` and select **Enable loadable module support**. The kernel is configured to use the memory starting after the first 256MB as the memory is segmented so that FreeRTOS has the first 256MB of memory and Linux has the rest.

```
Kernel Configuration --->
    (0x10000000) Physical address of main memory
    [*] Enable loadable module support  --->
```

3. Select **High Memory Support** within **Kernel Features**

4. 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
```

5. Enable **Userspace firmware loading support** and configure external firmware blobs:

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

TIP: *Please note that if you want to use Zynq `remoteproc` driver as a built-in driver, you will need to enable **Include in-kernel firmware blobs in kernel binary**, as the firmware must be available during driver initialization, which is before the root file system is available.*

- The **External firmware blobs to build into the kernel binary** is the file name of the FreeRTOS firmware.
- **Firmware blobs root directory** is the directory name in the root filesystem which contains the firmware. In this case, the directory to hold the firmware in the target root filesystem is "`/lib/firmware`".

6. 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
```

You can enable the the drivers as a Module **<M>** or Built-in **<*>**. Exit the menuconfig and save changes.

System Configuration

1. Configure the PetaLinux System Settings to support AMP as follows:

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

2. Configure the system to boot using the **old Uboot kernel image** format. This step is required due to an issue with U-Boot and kernel start memory addresses which are non-zero, see *U-Boot Old Kernel Image* for more information.

```
PetaLinux Configuration --->
    System Settings --->
        [*] Build old Uboot kernel image
```

3. Select the Linux AMP demo application. The **latencystat** demo can be found in the **PetaLogix Demo Applications** submenu.

```
PetaLinux Configuration --->
    PetaLogix Demo Applications  --->
        [*] latencystat  --->
```

DTS (Device Tree Source) Setup

The PetaLinux 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 BSP is generated from a hardware description and requires modification to support AMP.
Open the DTS file (with a text editor) located in the PetaLinux kernel directory, where `vendor` and `product` are the respective names of your AMP platform created in the previous sections:

```
"$PETALINUX/software/linux-2.6.x/arch/arm/boot/dts/<vendor>-<product>.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 0 0 38 0 >;
        firmware = "freertos";
        ipino = <5>;
        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 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 (built-in)

IMPORTANT: *This section only applies if the `remoteproc` and associated drivers are set as built-in.*

1. The kernel must have access to the firmware image during its build, a copy of the FreeRTOS application ELF file must exist at "`linux-2.6.x/firmware`".

```
$ cd <Path-to-XSDK-workspace>/<FreeRTOS-application-directory/Debug
$ cp <FreeRTOS-application>.elf \
  $PETALINUX/software/linux-2.6.x/firmware/<firmware-name>
```

WARNING: ***firmware-name*** *is the string that is set in the kernel config as* ***External firmware blobs to build into the kernel binary****.*

2. Run make inside the "`$PETALINUX/software/petalinux-dist`" directory:

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

Install the Firmware Image (module)

IMPORTANT: *This section only applies if the `remoteproc` and associated drivers are set as module.*

The firmware image for the FreeRTOS application must be placed in the firmware directory of the root filesystem "`$PETALINUX/software/petalinux-dist/romfs/lib/firmware/`".

```
$ mkdir $PETALINUX/software/petalinux-dist/romfs/lib/firmware
$ cd <Path-to-XSDK-workspace>/<FreeRTOS-application-directory/Debug
$ cp <FreeRTOS-application>.elf \
  ${PETALINUX}/software/petalinux-dist/romfs/lib/firmware/<firmware-name>
```

Build PetaLinux

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

Create the BOOT.BIN

The generated kernel image and u-boot images are in the "`$PETALINUX/software/petalinux-dist/images`" directory. In order to create a boot image, use the `petalinux-gen-zynq-boot` to generate the "`BOOT.BIN`".

```
$ petalinux-gen-zynq-boot -b <path-to-FSBL> -f <path-to-FPGA-bitstream>
```

The "`BOOT.BIN`" file will be in the "`$PETALINUX/software/petalinux-dist/images`" directory. Now that all images are created you can boot the system.

Known Issues

U-Boot Old Kernel Image

By default, PetaLinux will use the FIT image format for the kernel image, but the U-Boot in PetaLinux v2013.04 is unabled to load the kernel image if the physical address of the main memory is set to a non-zero value.

This will be resolved in a future release. Currently we will use the old Uboot kernel image format for the kernel image.

Trace Buffer is slow

The trace buffer is slow due to the CPU cache which has a delayed writeback.

This will be resolved in a future release. This does not affect functionality and the trace buffer can still be accessed. Alternatively the FreeRTOS application can use a UART to display log information.

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)