

PetaLinux SDK User Guide

Zynq AMP Linux FreeRTOS Guide

UG978 (v2012.12) December 17, 2012



Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

Date	Version	Notes
2012-12-17	2012.12	Initial Release for PetaLinux v2012.12 and Xilinx 14.4

Table of Contents

Revision History	1
Table of Contents	2
About this Guide	3
Related PetaLinux Documents	3
Prerequisites	3
Overview	4
FreeRTOS Demo Application	5
Linux Demo Application	5
Installation	6
Testing Pre-Built Reference Design	7
Boot PetaLinux	7
Starting FreeRTOS Firmware	7
Demo Application	8
Accessing the Trace Buffer	9
Bringup a Linux FreeRTOS AMP System on the ZC702	11
Hardware Setup	11
XSDK Setup	12
Setup Workspace	12
Create FSBL	12
Create PetaLinux BSP	13
Create FreeRTOS BSP	14
Create FreeRTOS Application	16
PetaLinux Configuration	18
Platform Setup	18
Kernel Configuration	18
System Configuration	19
DTS (Device Tree Source) Setup	20
Build PetaLinux with AMP support	21
Install the Firmware Image (built-in)	21
Install the Firmware Image (module)	21
Build PetaLinux	21
Create the BOOT.BIN	22
Known Issues	23
Kernel High Memory Support	23
U-Boot Old Kernel Image	23
Trace Buffer is slow	23

About this Guide

This document 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 pre-built guide
- 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.

Related PetaLinux Documents

The following other documents exist to help you to make the most of your PetaLinux experience:

- Getting Started Guide
- Application Development Guide
- QEMU System Simulation Guide
- Board Bringup Guide

Prerequisites

This document assumes that the following prerequisites have been satisfied:

- PetaLinux SDK has been installed.
- You know how to build a PetaLinux software image.
- You know how to boot a PetaLinux software image.
- PetaLinux setup script has been sourced in each command console with 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.

Overview

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

The Linux-FreeRTOS AMP system described is designed to demonstrate Linux's ability to configure both the Linux kernel, as well as the configuration of the secondary CPU for FreeRTOS with the loading of the 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 CPU to CPU communication. This is implemented with pre-allocated memory segments which are used for the passing 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 use of 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.

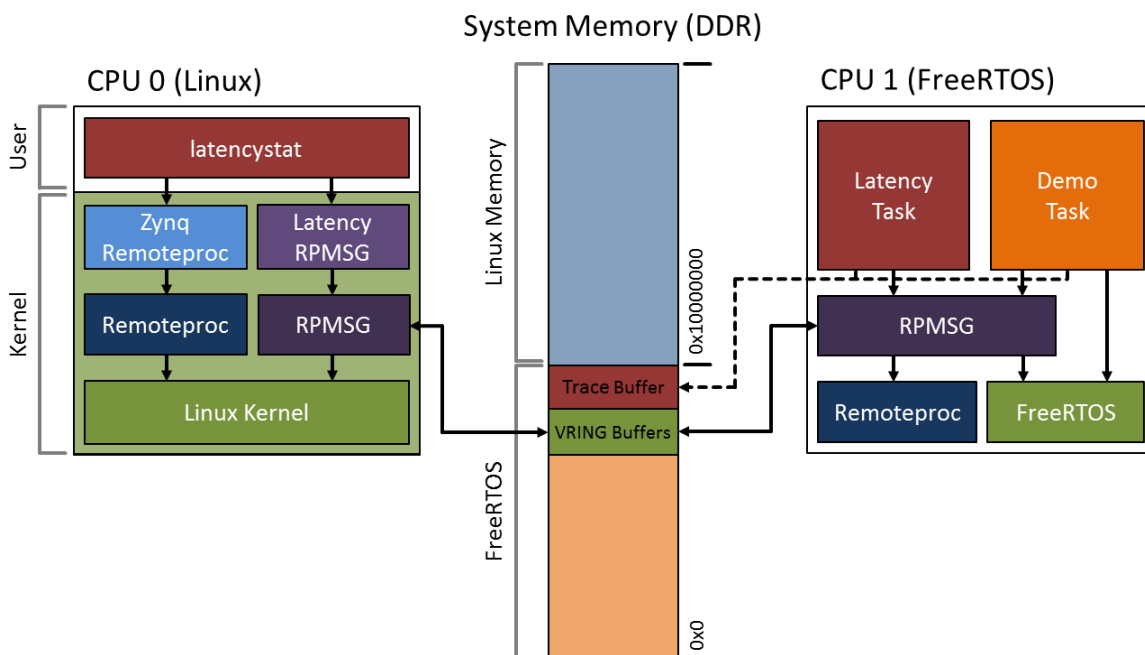


Figure 1: Linux-FreeRTOS AMP Reference Design

FreeRTOS Demo Application

The demo application provided in the reference design demonstrates the use of the `rpmmsg` 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 `rpmmsg` 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-14.4.bsp
```

After you have installed the BSP, you will find the ZC702 AMP reference design from "\$PETALINUX/hardware/reference-designs/Xilinx-ZC702-AMP-14.4/" 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-14.4
 - 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

- Configure the ZC702 to use SD boot mode, that is connect 1-2 of jumper J22 and J25 on the board.
- Connect the UART port on ZC702 to your host
- Connect the Ethernet port on ZC702 to a local network
- Copy the `B00T.BIN` and `image.ub` from the pre-built images directory: `Xilinx-ZC702-AMP-14.4/pre-built/images/` to a SD card.
- Insert the SD card into the SD card slot on ZC702 and then power on the board.
- 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

- Watch the console. Log into PetaLinux with username: "root" and password: "root".
- 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
```

- 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
```

- Load the 2nd processor with FreeRTOS firmware as follows:

```
# modprobe zynq_remoteproc
```

- 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

- 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. To run the `latencystat` demo linux userspace application to get the FreeRTOS interrupt latency data, 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
```

- Run `latencystat` demo application as follows:

```
# latencystat -b
```

- 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 then reply with the histogram data, and then the application dumps the data.

The `latencystat` demo application can also display the information in a graph format or dump the data directly 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 by the FreeRTOS application. This Trace Buffer can be used as a logging console to display 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. For reference the default Trace Buffer is 32 KB in size.

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

```
/debugfs/remoteproc/remoteproc0/trace0
```

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

```
# cat /debugfs/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
rmsg: CLEAR request
rmsg: 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_demo: task resumed as expected
rmsg: STOP request
rmsg: CLONE request
rmsg: GET request
rmsg: QUIT request
```



IMPORTANT: *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 "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 *"Board Bringup Guide"* 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.



IMPORTANT: *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 ps7_uart1 which is configured for use by Linux, select ps7_uart0 for use by FreeRTOS.

The FreeRTOS Demo Application provided uses the second Triple Timer Counter (TTC1), enable the "Timer 1" and set its IO as EMIO.

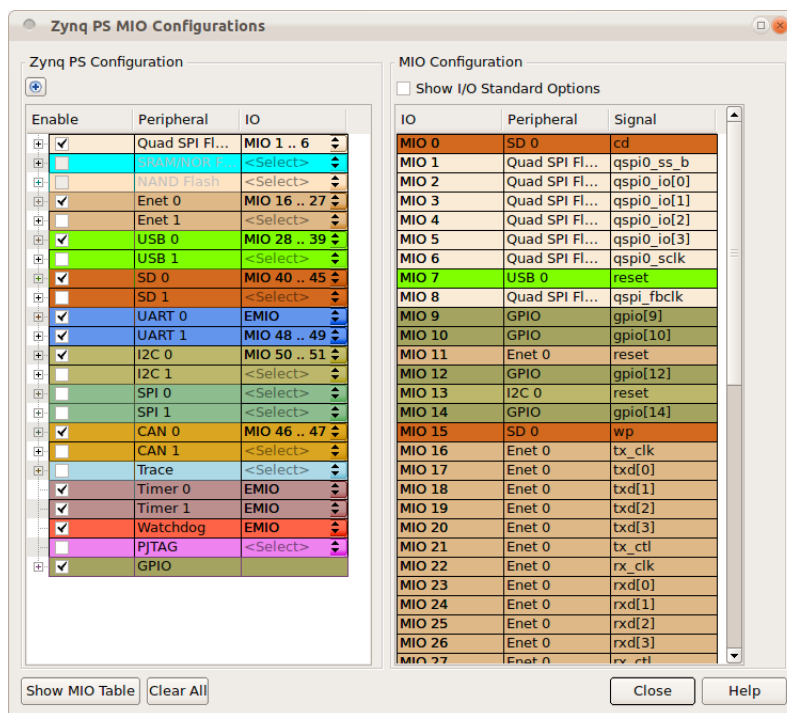


Figure 2: Zynq Peripheral Configuration

Once the project is configured, be sure to "Export Design to XSDK" from XPS to update the hardware description files of XSDK workspace once you update your hardware project.

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:



IMPORTANT: *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")*

- 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"
- Click "Rescan Repositories"
- Click "Apply"
- Click "OK"

Create FSBL

Create a Xilinx FSBL application project as follows:

- Go to File > New > Application Project
- Select "ps7_cortexa9_0" as "Processor" from the New Project wizard
- Select "Zynq FSBL" as the "Select Project Template"
- Click "Next" and then "Finish" to create the project

Create PetaLinux BSP

Create a PetaLinux BSP as follows:

- Go to File > New > Board Support Package
- Select "ps7_cortexa9_0" as the "CPU" in the "New Board Support Package Project" wizard
- Select "petalinux" as "Board Support Package OS"

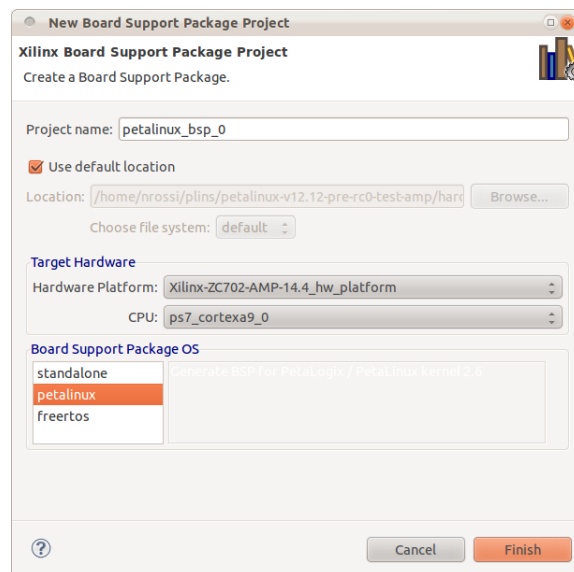


Figure 3: New BSP Wizard - PetaLinux

- Click "Finish". A "Board Support Package Settings" window will pop up.
- Select "petalinux" from the "Overview" in the "Board Support Package Settings" window
- Select "ps7_uart_1" as "stdout" and "stdin" in the "Configuration for OS table"
- Select "ps7_ddr_0" as the "main_memory"
- Select "ps7_qspi_0" as the "flash_memory"
- Select "ps7_sd_0" as the "sdio"
- Select "ps7_ethernet_0" as the "ethernet"

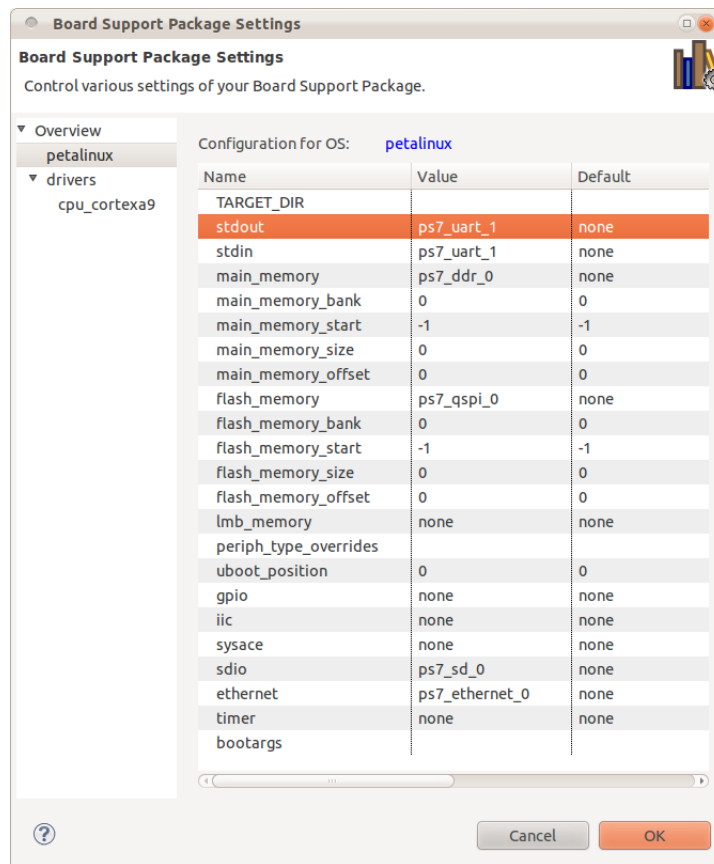


Figure 4: PetaLinux BSP Configuration

- Click "OK"

Create FreeRTOS BSP

Create a FreeRTOS BSP as follows:

- Go to File > New > Board Support Package
- Select "ps7_cortexa9_1" as the "CPU" in the "New Board Support Package Project" wizard
- Select "freertos" as "Board Support Package OS"

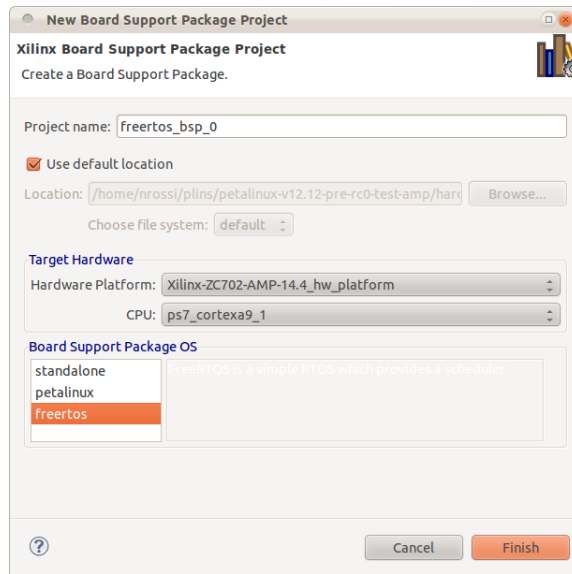


Figure 5: New BSP Wizard - FreeRTOS

- Click "Finish". A "Board Support Package Settings" window will pop up.
- Select "freertos" from the "Overview" in the "Board Support Package Settings" window
- Select "ps7_uart_0" as "stdout" and "stdin" in the "Configuration for OS table"

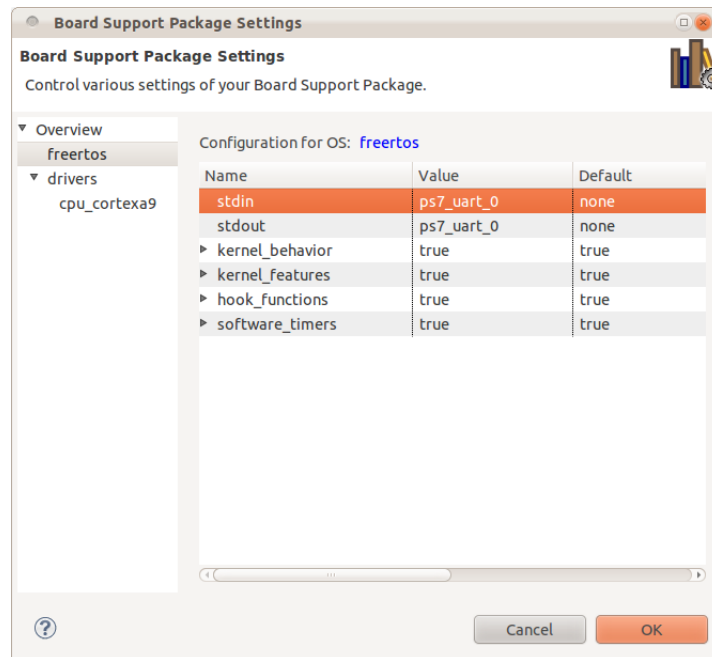


Figure 6: FreeRTOS BSP Configuration

- Select "cpu_cortexa9" from Overview > drivers

- Add "-DUSE_AMP=1" to the "extra_compiler_flags" of the "Configuration for driver" table.

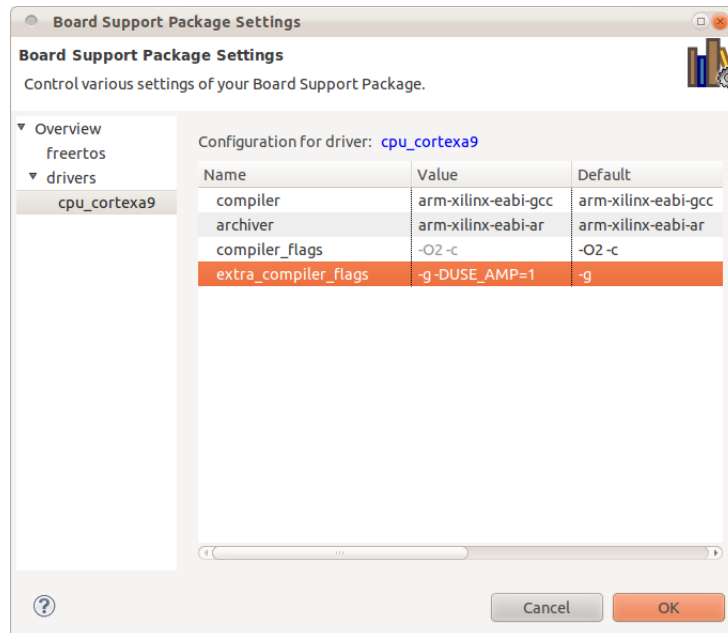


Figure 7: FreeRTOS BSP Configuration

- Click "OK"

Create FreeRTOS Application

Create a FreeRTOS AMP application project as follows:

- Go to File > New > Application Project
- 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.

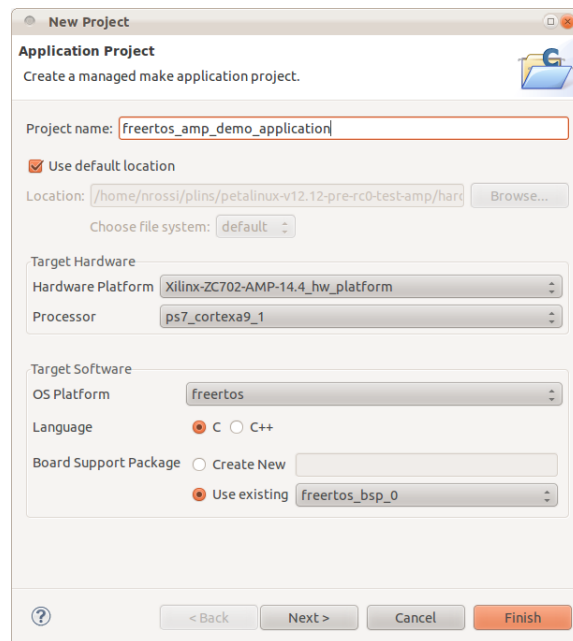


Figure 8: New Application Project - FreeRTOS Demo Application

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

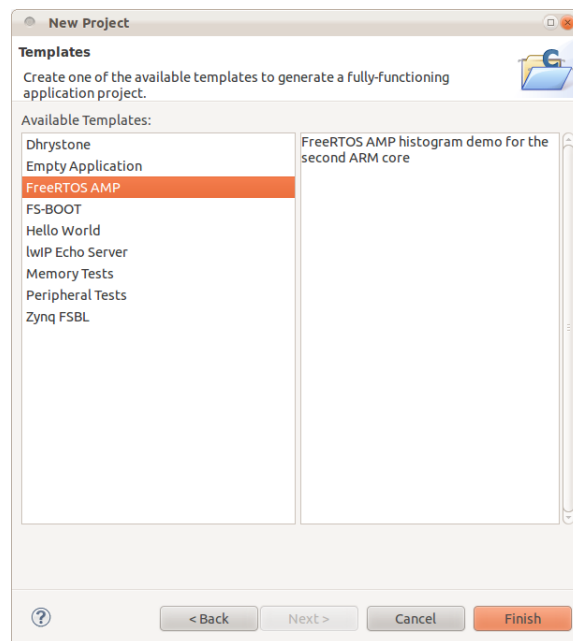


Figure 9: New Application Project - FreeRTOS Demo Application

- 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 PetaLinux tool as follows, specify a vendor name and platform name to use when creating the platform.

```
$ petalinux-new-platform -c arm -v Xilinx -p ZC702-AMP-14.4
```

The `petalinux-new-platform` will create and select the new platform.

Kernel Configuration

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

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

- Set "Physical address of main memory" to "0x10000000" and enable "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 --->
```

- Disable "High Memory Support", within "Kernel Features". This step is required due to an issue with High Memory support and AMP see "*Kernel High Memory Support*" for more information.

```
Kernel Configuration --->
  Kernel Features --->
    [ ] High Memory Support
```

- 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
```

- The "External firmware blobs to build into the kernel binary" is file name of the FreeRTOS firmware.



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.

- o "Firmware blobs root directory" is the name of directory in root filesystem which contains the firmware. In this case, the directory to hold the firmware in the target root filesystem is /lib/firmware.

- Enable Remoteproc and Rpmmsg drivers:

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

You can enable the the drivers as a Module <M> or Built-in <*>.

System Configuration

- Configure the PetaLinux System Settings to support AMP as follows:

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

- Configure the system to boot using to use "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
```

- 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, 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"
```

- FreeRTOS and Linux cannot share memory space, in order to allow both systems to function the memory is split into two sections. FreeRTOS is located in the lower memory segment starting at address 0x0.

Go to the "ps_dds_0: memory" device node, change the "reg" property value to "0x0 0x10000000 0x10000000 0x20000000".

```
ps7_dds_0: memory@0 {
    device_type = "memory";
    reg = <0x0 0x10000000 0x10000000 0x20000000>
};
```

Where the first set of numbers are the FreeRTOS memory segment (starting at 0x0 and has a size of 0x10000000). The second set represents the Linux kernels memory segment (starting after the FreeRTOS segment at 0x10000000).



IMPORTANT: Please note that the size of the kernel segment is limited to 0x20000000 (512 MB) as High memory support is disabled in the kernel.

- 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 requires matching 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" is the IPI from firmware to Linux,
- "vring0" and "vring1" 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)



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

- The kernel must have access to the firmware image during its build, a copy of the 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>
```



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

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

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

Install the Firmware Image (module)



RECOMMENDED: *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

Kernel High Memory Support

Kernel High Memory Support is not functional when using the AMP kernel configuration.

In order to boot the kernel and load the remoteproc drivers the High Memory Support must be disabled.

U-Boot Old Kernel Image

By default, PetaLinux will use the FIT image format for the kernel image, but the U-Boot in PetaLinux v2012.12 has issues when loading the kernel image if the physical address of the main memory is set to a non-zero value.

This issue will be resolved in a future release, currently, as a workaround, we will use the old Uboot kernel image format for the kernel image.

Trace Buffer is slow

This is a known issue with the trace buffer, it is caused by the CPU cache which has a delayed writeback.

This issue will be resolved in a future release, currently, 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.