OpenAMP Framework Getting Started Guide

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/doc_feedback_form

Table of Contents

List of Figures

List of Tables

Chapter 1
OpenAMP Framework Overview

This document presents the steps to setup the environment, build and execute the sample applications, tests, and firmware provided within the OpenAMP Framework package.

This overview contains the following information for getting started with the OpenAMP Framework:

OpenAMP Framework Configuration

The OpenAMP Framework package works with multiple configurations.

Table 1-1 shows a summary of the these configurations and the capabilities showcased in these configurations:

Table 1-1. OpenAMP Framework Configurations

Config	Master	Remote	Capabilities showcased using OpenAMP Framework
1	Linux	bare metal	• Ability to load a remote bare metal context from the Linux master. • Ability to perform IPC from the Linux master to the bare metal remote.
2	bare metal	Linux	• Ability to load a remote Linux context from the bare metal master. • Ability to perform IPC from the Linux rpmsg master to the bare metal rpmsg remote.

Related Topics

Supported Environment

Directory Structure

Supported Environment

The OpenAMP Framework is supported for the following environment:

Table 1-2 shows the hardware platform, Linux version, and the toolset.

Table 1-2. Supported Environments

Hardware Platform	Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit
Linux	PetaLinux v2013.10
Toolset	The OpenAMP Framework SW configurations were tested with two tool chains: • Mentor Graphics CodeSourcery GNU Lite tools: • Target: arm-non-eabi, gcc version 4.8.1 (Sourcery CodeBench Lite 2013.11-24) • Xilinx GCC tools: • Target: arm-xilinx-eabi, gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-39)

Related Topics

OpenAMP Framework Configuration Supported Environment

Directory Structure

The directory structure looks like below:

```
+   ---- apps                 -----------> OpenAMP Framework applications
¦    + ---- samples           -----------> OpenAMP Framework Samples
¦          + ---- master      -----------> Master sample applications
¦          + ---- remote      -----------> Remote sample applications
¦    + ---- tests             -----------> OpenAMP Framework Tests
¦          + ---- master      -----------> Master test applications
¦          + ---- remote      -----------> Remote test applications
+ ---- common                 -----------> Common OpenAMP Framework components
+ ---- docs                   -----------> Documentation
+ ---- include                -----------> External include files
¦    + ---- open_amp.h
+ ---- libs                  ------------> OpenAMP Framework lib and System libs for BME.
+ ---- porting
```

```
¦    + ---- config              ------------> Interface to define system level configuration
                                              information like-remote firmware info.
¦    + ---- env                 ------------> Software environment interface (OS or bare metal)
¦    + ---- hil                 ------------> Hardware interface layer
+ ---- proxy                    ------------> Proxy infrastructure for Linux master and bare metal
remote
 + ---- remoteproc              -------------> remoteproc sub-component
 + ---- rpmsg                   -------------> rpmsg sub-component
 + ---- virtio                  -------------> virtio sub-component
```

Related Topics

OpenAMP Framework Configuration
Supported Environment

Chapter 2
OpenAMP Framework Applications and Tests

This chapter briefly describes the sample applications and tests provided within the OpenAMP Framework package for each supported system configuration.

Linux Master/Bare Metal Remote Configuration

This demonstration showcases the Life Cycle Management (LCM) of the remote processor and its software context and the Inter Processor Communication (IPC) with remote software context.

Matrix Multiplication Demonstration

The Linux kernels space application on the master processor showcases IPC with remote bare metal based firmware using rpmsg. The zynq_remoteproc_driver is used to load the remote firmware image.

The Linux user space application performs the same matrix multiplication functionality by accessing the rpmsg character device created and handled by the rpmsg_user_dev_driver.

1. Linux master boots remote bare metal based firmware using the zynq_remoteproc_driver.
2. Linux master application transmits two random matrices to remote firmware using rpmsg.
3. Bare metal based firmware performs multiplication and transmits results back to master using rpmsg.
4. Linux master application prints the results of the matrix multiplication

Application	**Location**
Linux Master (kernel space)	*apps/samples/master/linux/kernelspace/rpmsg_mat_mul_kern_app*
Linux Master (user space)	*apps/samples/master/linux/userspace/matrix_multiply*
bare metal remote firmware	*apps/samples/remote/baremetal/matrix_multiply*

Echo Test Application

Description

This application tests data integrity of the payload transmitted. The Linux master loads bare metal based firmware first, executes the echo test and performs shutdown.

Echo test between Linux master<-> Baremetal remote based firmware

1. Linux master boots remote bare metal based firmware using remoteproc.
2. Linux master application transmits echo payloads to remote firmware using rpmsg.
3. Bare metal remote based firmware echoes back the payload received to the master using rpmsg.
4. Linux master application verifies integrity of payload received and prints results.

Application	**Location**
Linux master (kernel space)	*apps/tests/master/linux/kernelspace/rpmsg_echo_test_kern_app*
Linux master (user space)	*apps/tests/master/linux/userspace/echo_test*
bare metal remote	*apps/tests/remote/baremetal/echo_test*

Proxy Application

Description

This application allows the firmware on the remote core to use console and execute file I/O on master by communicating with a proxy application running on the Linux master.

Demonstration to showcase proxy infrastructure for Linux master <-> Baremetal remote configuration.

1. The Linux master boots bare metal based remote firmware using proxy_app.
2. Remote firmware does File I/O on Linux FS on master processor, and uses master console to receive input from the user and display output.

Application	**Location**
Linux master (kernel space driver)	*proxy/master/linux/kernelspace*
Linux master (user space application)	*proxy/master/linux/userspace*
bare metal remote	*apps/samples/remote/baremetal/rpc_demo*

Related Topics

Bare Metal Master /Linux Remote Configuration

Bare Metal Master /Linux Remote Configuration

This demonstration showcases the Life Cycle Management (LCM) of the remote processor and its software context and the Inter Processor Communication (IPC) with remote software context.

Matrix Multiplication Demonstration

Description

Bare metal based master application uses remoteproc to boot and shut down Linux on a remote processor. Once Linux is booted on the remote processor, the matrix multiplication demo can be executed that showcases IPC between remote and master.

To demonstrate LCM and IPC between Baremetal master<->Linux remote based firmware

1. The bare metal based master application boots Linux on remote processor using remoteproc.
2. The Linux application on remote processor acts as rpmsg master and transmits random matrices to bare metal using rpmsg.
3. The bare metal application on master core performs multiplications and transmits results back to Linux using rpmsg.
4. Linux application on remote core prints the results of matrix multiplication.
5. User selects quit from Linux application, Linux sends a shutdown message to bare metal master.
6. Linux executes a system halt call to gracefully shutdown itself.
7. Bare metal on master core shuts down the remote core

Application	Location
bare metal master	*apps/samples/master/baremetal/matrix_multiply*
Linux rpmsg master (kernel space)	*apps/samples/master/linux/kernelspace/rpmsg_mat_mul_kern_app*
Linux rpmsg master (user space)	*apps/samples/master/linux/userspace/matrix_multiply*

Echo Test Application

Description

This application tests the data integrity of the payload transmitted. The bare metal master application loads Linux based firmware first, executes the echo test and performs shutdown.

Echo test between Baremetal master<->Linux remote based firmware

1. The bare metal master based application boots Linux remote based firmware using remoteproc.
2. Linux remote firmware acts as rpmsg master and transmits echo payloads to bare metal using rpmsg.
3. Bare metal echoes back the payload received to Linux using rpmsg.
4. Linux based application verifies integrity of payload received and prints the results.
5. User selects quit from the Linux application, Linux sends a shutdown message to remoteproc bare metal master.
6. Linux executes a system halt call to gracefully shutdown itself.
7. Bare metal on master core shuts down the remote core.

Application	**Location**
bare metal master	*apps/tests/master/baremetal/echo_test*
Linux rpmsg master (kernel space)	*apps/tests/master/linux/kernelspace/rpmsg_echo_test_kern_app*
Linux rpmsg master (user space)	*apps/tests/master/linux/userspace/echo_test*

Function Test Suite Application

Description

This application performs functional tests on various features and capabilities supported by the OpenAMP Framework. The following capabilities are tested:

- rpmsg send, and send_off channel
- remote channel deletion
- endpoint creation and deletion
- multiple remote firmware life cycles from a single master run-time instance

The bare metal master application loads Linux based firmware first, executes functional tests and performs shutdown.

Functional tests between Baremetal master<-> Linux remote based firmware

1. The bare metal master based application boots Linux remote based functional test firmware using remoteproc.
2. The bare metal master based application conducts functional tests with Linux firmware and sends results to Linux which prints the results on console

Application	Location
bare metal master	*apps/tests/master/baremetal/func_test_suite*
Linux rpmsg master (kernel space)	*apps/tests/master/linux/kernelspace/rpmsg_func_test_kern_app*

Related Topics

Linux Master/Bare Metal Remote Configuration

Chapter 3
Environment Setup, Build, and Execution of Applications and Test Binaries

This chapter contains instructions for environment setup, building and executing applications and test binaries.

OpenAMP Framework Environment Setup

Here you will find a description of the toolset, build environment, installation, and running the setup script.

Toolset

The SW configurations were tested with the tool-chains listed in Table 3-1:

Table 3-1. OpenAMP Framework Environment

Supported Tool chains	**Notes**
Mentor Graphics CodeSourcery Lite tools Target: arm-none-eabi, gcc version 4.8.1 (Sourcery CodeBench Lite 2013.11-24)	A free version of Sourcery CodeBench tools can be obtained from `http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/`
Xilinx GCC Tools Target: arm_xilinx-eabi, gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-39)	

OpenAMP Framework Environment

Assume that the OpenAMP Framework package is installed at location <open_amp_root>.

To setup the environment for Mentor Graphics CodeSourcery GNU Lite tools - Target: arm-none-eabi, gcc version 4.8.1 (Sourcery CodeBench Lite 2013.11-24).

```
> $ PATH=<Mentor Graphics Sourcery CodeBench Lite for ARM EABI
Installation  Path>/bin:${PATH}
> cd <open_amp_root>
```

Modify the CROSS parameter in *Makefile.commons* to look like this

```
CROSS := arm-none-eabi-
```

To setup the environment for Xilinx GCC tools - Xilinx GCC tools - Target: arm-xilinx-eabi, gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-39)

```
> $ PATH="<Xilinx ARM EABI Installation Path>/bin:${PATH}
> cd <open_amp_root>
```

Modify CROSS parameter *Makefile.commons* to look like this

```
CROSS := arm-xilinx-eabi-
```

Environment variable

- Change directory to <*open_amp_root*>

  ```
  $ export OPENAMP=$PWD
  ```

Note

The scope of the variable $OPENAMP is local to the current shell. You must define it whenever you start working with a new shell terminal.

Related Topics

Setting up the PetaLinux Environment

Setting up the PetaLinux Environment

In order to become familiar with PetaLinux workflows and the environment setup guidelines, you are advised to go through the following documents:

`PetaLinux SDK User Guide: Installation Guide (ver 2013.10)`

`PetaLinux SDK User Guide: Zynq AM Linux FreeRTOS (ver 2013.10)`

`PetaLinux SDK User Guide: Application Development Guide (ver 2013.10)`

`PetaLinux SDK User Guide: Getting Started Guide (ver 2013.10)`

The following section walks you through the steps required to setup a PetaLinux environment.

Prerequisites

The environment must satisfy the following requirements as per PetaLinux SDK User Guide, Installation Guide:

- Minimum workstation requirements:
  - 2GB RAM (recommended minimum for Xilinx tools)
  - Pentium 4 2GHz CPU clock or equivalent

- 5 GB free HDD space
- Supported OS:
  - RHEL 5 (32-bit or 64-bit)
  - RHEL 6 (32-bit or 64-bit)
  - SUSE Enterprise 11 (32-bit or 64-bit)

- PetaLinux release package downloaded.
- Valid PetaLinux license.

Note

You can also setup a PetaLinux environment on Ubuntu 12.04 LTS using a bash shell.

The following documentation also assumes that you are familiar with basic Linux environment, commands and shell routines.

For Ubuntu 12.04LTS, please make sure that the following tools and libraries have been installed:

Table 3-2. Ubuntu Tools and Libraries

Tools/Library	ATP Package for Ubuntu
dos2unix	tofrodos
ip	iproute
gawk	gawk
gcc	gcc
git	git-core
gpg	gnupg
make	Make
netstat	net-tools
ncurses	ncurses-dev
tftp server	tftpd
zlib	zliblg-dev
flex	flex
bison	bison

Procedure

1. Please refer to the to

`PetaLinux SDK User Guide: Installation Guide (ver 2013.10)`

for PetaLinux and license installation.

2. Invoke the Petalinux installer with the following command:

   ```
   $ ./petalinux-v2013.10-final-installer.run <installation root>
   ```

   A successful execution installs the PetaLinux in a directory named *petalinux-v2013.10-final* within the current working directory.

3. Run Setup Script

   You must source the appropriate setup script.

   - For bash shell:

     ```
     $ source <path-to-installed-PetaLinux>/settings.sh
     ```

   - For C shell:

     ```
     source <path-to-installed-PetaLinux>/settings.csh
     ```

Results

Executing these commands displays an output similar to the following:

```
PetaLinux environment set to '<installation path>/petalinux-v2013.10-
final'
INFO: Finalising PetaLinux installation
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other service
```

Verify that the PetaLinux environment has been correctly setup:

```
$ echo $PETALINUX
<installation path>/petalinux-v2013.10-final
```

Note
You are required to run a setup script each time a shell terminal is started for PetaLinux build.

Note
In certain cases, the shell could be linked to bin/dash instead of bin/bash as default and will result in error messages.

To resolve this issue, link bin/sh to bin/bash:

```
$ sudo rm /bin/sh
```

```
$ sudo ln -s /bin/bash /bin/sh
```

Related Topics

OpenAMP Framework Environment Setup

Setting up the Hardware

This section details the hardware setup to execute an OpenAMP Framework application. Execute these steps and repeat the necessary steps (copying images to SD card) in order to execute the applications as described in the following sections.

Procedure

1. Power off the board.
2. Connect the serial port on the board (USB UART Connector J17) to your workstation using a USB mini-B cable.
3. Set the switches on the board to boot from the SD card:

SW16.1	SW16.2	SW16.3	SW16.4	SW16.5
0	0	1	1	0

4. Copy the U-Boot and application binaries (as described in the following sections) on an FAT32 formatted SD card.
5. Place the SD card into the J64 slot.
6. Power up the board.
7. Open a serial terminal on your workstation with the following settings:

   baud rate = 115,200

   data bits = 8

   stop bits = 1

   flow control = none
8. Wait for the U-Boot console to appear on the serial terminal.

Note
It is recommended to format the SD card with a single FAT32 partition 0.

Related Topics

OpenAMP Framework Environment Setup

Build and Execute Applications for Linux Master/Baremetal Remote Configurations

In order to build and run applications in this configuration, you need to first build remote bare metal applications and then install these firmware images within the Linux master kernel root file system.

Building the OpenAMP Framework Library, Applications, and Tests

Follow these steps to build the OpenAMP Framework library, applications and/or tests.

Prerequisites

- OpenAMP Framework library installed at location $OPENAMP

Procedure

1. Change directory to $OPENAMP

   ```
   $ cd $OPENAMP
   ```

2. Execute the build script. This operation will build remote applications for the bare metal environment.

   Note: The build operation will generate application outputs in the respective application directories.

   ```
   $ source open_amp_build.sh
   ```

3. If you need to clean all previous builds, you can execute the script with the following option:

   ```
   $ source open_amp_build.sh - c
   ```

Related Topics

Building Linux Master Applications and Tests   Executing Applications and Tests for Linux Master/ Baremetal Remote Configurations

Building Linux Master Applications and Tests

The following steps are required to build PetaLinux and Linux applications for master Linux kernel.

Procedure

1. Create a project for the Linux Master:

   a. Change the directory to the location where the Linux Master project needs to be created and execute the following command:

   ```
   >petalinux-create -t project -n <master_name> --template zynq
   ```

   <*master_name*> is the user defined name of the Linux Master project. The location of project will be referred as <*master-root*> from this point onwards.

2. Navigate to <*master-root*> and enable the following configuration options in kernel.

   a. Set up PetaLinux so that the Linux kernel starts at address 0x10000000.

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

      i. Change the Kernel base address value to 0x10000000 if it is not already set to this value.

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

      ii. Save the configuration changes.

   b. Execute the following command while in the <*master-root*> directory

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

      i. Select Enable loadable module support.

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

      ii. Select High Memory Support within Kernel Features

      iii. 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
```

iv. 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
```

v. Enable Remoteproc Driver

```
Kernel Configuration --->
   Device Drivers --->
      Remoteproc drivers (EXPERIMENTAL) --->
      <M> Support ZYNQ remote proc
```

Note

Enable the driver as a Module <M>

3. Set up Device TreeSource (DTS)

   a. Open DTS file with a text editor:

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

   b. Add the following device node for the Zynq remoteproc driver so that the driver can be probed.

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

4. Execute the automation scripts provided within $OPENAMP to create the necessary application projects, kernel driver projects, and firmware projects for master Linux project. The script creates projects, moves sources and make files from $OPENAMP as needed, enables the project to be included/enabled in the PetaLinux initramfs.

   ```
   >cd <master_root>
   ```

Note

OpenAMP Framework library and applications should be created using instructions under"Building the OpenAMP Framework Library, Applications, and Tests" on page 22 before proceeding beyond this point.

```
>source $OPENAMP/libs/system/zc702evk/linux/scripts/
         open_amp_create_projects.sh master <master-root>
```

where <master_root> is the path to the master Linux project.

It will generate the following user-space application and kernel modules:

```
1. <master_root>/components/apps/mat_mul_demo
2. <master_root>/components/apps/echo_test
3. <master_root>/components/apps/proxy_app
4. <master_root>/components/modules/zynq_remoteproc_driver
5. <master_root>/components/modules/rpmsg_mat_mul_kern_app
6. <master_root>/components/modules/rpmsg_echo_test_kern_app
7. <master_root>/components/modules/rpmsg_user_dev_driver
8. <master_root>/components/modules/rpmsg_proxy_dev_driver
9.<master_root>/components/apps/mat_mul_baremetal_fw
10.<master_root>/components/apps/echo_test_baremetal_fw
11.<master_root>/components/apps/rpc_demo_baremetal_fw
```

5. Build PetaLinux

   a. Change into the <master_root>

   ```
   $ cd <master_root>
   ```

   b. Execute the following command:

   ```
   $ petalinux-build
   ```

Results

A successful build will generate a Linux FIT image (*image.ub*) at:

```
<master_root>/images/linux
```

Related Topics

Building the OpenAMP Framework Library, Applications, and Tests

Executing Applications and Tests for Linux Master/ Baremetal Remote Configurations

Executing Applications and Tests for Linux Master/ Baremetal Remote Configurations

This is a procedure for moving the U-Boot, application and test binaries to boot medium (SD Card).

Procedure

1. Place the U-Boot image for ZC702EVK(*<master_root>/pre-built/linux/images/BOOT.bin*) and PetaLinux FIT image (*<master_root>/images/linux/image.ub*) in an SDCARD formatted for FAT.
2. Insert the SDCARD in the SDIO slot on ZC702EVK.
3. Power up board to boot U-Boot
4. In the U-Boot console:
    - fatload mmc <dev>:<part> 0x1000000 image.ub
    - bootm 0x1000000
5. PetaLinux boots up
6. Username:root, Password:root

    NOTE:

    <dev> refers to the relevant MMC device number. The list of available devices can be retrieved from the U-boot prompt by entering - U-Boot-PetaLinux> mmc list

    <part> refers to the relevant partition on the current mmc device. It can be retrieved from the U-Boot prompt by entering - U-Boot-PetaLinux> mmc part

Results

Once PetaLinux is up and you have performed the root login, you are ready to load and execute one of the firmwares present in the Linux root FS.

Related Topics

Building the OpenAMP Framework Library, Applications, and Tests

Building Linux Master Applications and Tests

Executing the Matrix Multiply Application

This section explains how one can load and execute samples and test applications built into the Linux FS.

On completion of execution of a given application or test, follow the instructions provided to shutdown and unload the relevant kernel drivers before executing the next application.

Procedure

1. Execute the Kernel Space Matrix Multiplication sample application.

    a. Load Zynq platform remoteproc module

- For bare metal based remote matrix multiply firmware:

```
modprobe zynq_remoteproc_driver firmware=
”/lib/firmware/zc702evk/baremetal/matrix_multiply/firmware”
```

b. Load rpmsg_mat_mul_kern_app module to execute the application

```
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_mat_mul_kern_app
```

The matrix multiply kernel-space application output should look similar to:

```
Demo Start - Demo rpmsg driver got probed
 since the rpmsg device associated with driver was found !

 Create endpoint and register rx callback

 Master : Linux : Generating random matrices

 Master : Linux : Input matrix 0

 1  0  0  5  4  8
 0  0  4  7  4  3
 5  4  7  2  5  4
 1  9  4  5  4  9
 2  8  3  7  5  5
 6  3  6  2  5  3

 Master : Linux : Input matrix 1

 6  8  8  4  9  1
 7  6  0  8  2  2
 7  5  4  4  5  3
 2  9  6  8  8  9
 9  2  8  9  7  1
 2  6  0  3  1  2

 Master : Linux : Sent 296 bytes of data over rpmsg channel to
remote

 Master : Linux : Received 148 bytes of data over rpmsg channel from
remote

 Master : Linux : Printing results
root@Xilinx-ZC702-2013_3:~#  68  109  70  104  85  66
 84  109  90  117  107  85
 164  151  120  153  143  65
 161  189  86  195  124  98
 158  182  110  200  145  105
 154  142  124  142  144  59
virtio_rpmsg_bus virtio0: destroying channel rpmsg-openamp-demo-
channel addr 0x1
```

You can now shutdown the firmware using the following set of commands:

```
root@Xilinx-ZC702-2013_3:~# modprobe -r rpmsg_mat_mul_kern_app
root@Xilinx-ZC702-2013_3:~# modprobe -r virtio_rpmsg_bus
root@Xilinx-ZC702-2013_3:~# modprobe -r zynq_remoteproc_driver
```

2. Execute the User Space Matrix Multiplication Application.

    a. Load Zynq platform remoteproc module

    - For bare metal based remote matrix multiply firmware:

    ```
    modprobe zynq_remoteproc_driver firmware=
    ”/lib/firmware/zc702evk/baremetal/matrix_multiply/firmware”
    ```

    b. Load the rpmsg user device driver using the following command:

    ```
    root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_user_dev_driver
    ```

    c. Start matrix_multiply user-space application

    ```
    root@Xilinx-ZC702-2013_3:~# mat_mul_demo
    ```

    The matrix multiply user-space application output should look similar to:

```
Matrix multiplication demo start

 Open rpmsg dev!

 Query internal info ..
 rpmsg kernel fifo size = 2048
 rpmsg kernel fifo free space = 2048

 Creating ui_thread and compute_thread ...

 ****************************************
 Please enter command and press enter key
 ****************************************
 1 - Generates random 6x6 matrices and transmits them to remote core
   over rpmsg ..
 2 - Quit this application ..
 CMD>1

 Compute thread unblocked ..
 The compute thread is now blocking on a read() from rpmsg device

 Generating random matrices now ...

 Master : Linux : Input matrix 0

 2  7  7  5  3  0
 7  1  9  4  5  3
 4  6  5  8  5  7
 4  1  8  0  1  9
 8  6  0  6  4  7
 5  9  4  2  4  0

 Master : Linux : Input matrix 1

 4  1  1  3  7  6
 8  3  4  6  3  0
 3  8  1  4  8  5
 3  8  1  3  6  8
```

```
 2   1   7   7   5   3
 7   9   6   0   4   4

 Writing generated matrices to rpmsg device, 296 bytes written ..

Received results! - 148 bytes from rpmsg device (transmitted from
remote context)

 Master : Linux : Printing results
 106  122  63  112  136  96
 106  146  77  110  185  146
 162  194  118  127  187  156
 113  153  77  57  136  103
 155  141  108  106  158  136
 118  84  75  119  126  78

 ****************************************
 Please enter command and press enter key
 ****************************************
 1 - Generates random 6x6 matrices and transmits them to remote core
over rpmsg ..
 2 - Quit this application ..
 CMD>
```

3. You can shutdown the firmware using the following set of commands:

```
root@Xilinx-ZC702-2013_3:~# modprobe -r rpmsg_user_dev_driver
root@Xilinx-ZC702-2013_3:~# modprobe -r virtio_rpmsg_bus
root@Xilinx-ZC702-2013_3:~# modprobe -r zynq_remoteproc_driver
```

Related Topics

Executing Applications and Tests for Linux Master/ Baremetal Remote Configurations

Executing the Echo Test Application

Executing the Echo Test Application

This procedure outlines the steps to execute the Linux user space or kernel space applications to perform echo test.

Procedure

1. Execute the Kernel Space Echo Application.
   a. Load Zynq platform remoteproc module
      - For bare metal based remote echo firmware:

        ```
        modprobe zynq_remoteproc_driver firmware=
        "/lib/firmware/zc702evk/baremetal/echo_test/firmware"
        ```

   b. Load rpmsg_echo_test_kern_app module to execute the application

```
root@Xilinx-ZC702-2013_3:~# modprobe virtio_rpmsg_bus
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_echo_test_kern_app
```

The echo test kernel-space application output should look similar to:

```
root@Xilinx-ZC702-AMP-2013_3:/# modprobe rpmsg_echo_test_kern_app
 remoteproc0: powering up 0.remoteproc-test
 remoteproc0: Booting fw image firmware, size 230628
 remoteproc0: remote processor 0.remoteproc-test is now up
virtio_rpmsg_bus virtio0: rpmsg host is online
virtio_rpmsg_bus virtio0: creating channel rpmsg-openamp-demo-
channel addr 0x1
 Echo Test Start!
 Master : Linux Kernal Space : Sending payload num 0 of size 9
 Master : Linux Kernal Space : Received payload num 0 of size 9
 Master : Linux Kernal Space : Sending payload num 1 of size 10
 Master : Linux Kernal Space : Received payload num 1 of size 10
 Master : Linux Kernal Space : Sending payload num 2 of size 11
 Master : Linux Kernal Space : Received payload num 2 of size 11
 Master : Linux Kernal Space : Sending payload num 3 of size 12
 Master : Linux Kernal Space : Received payload num 3 of size 12
 Master : Linux Kernal Space : Sending payload num 4 of size 13
 Master : Linux Kernal Space : Received payload num 4 of size 13
 Master : Linux Kernal Space : Sending payload num 5 of size 14
 Master : Linux Kernal Space : Received payload num 5 of size 14

 ...
 ...

 Master : Linux Kernal Space : Received payload num 483 of size 492
 Master : Linux Kernal Space : Sending payload num 484 of size 493
 Master : Linux Kernal Space : Received payload num 484 of size 493
 Master : Linux Kernal Space : Sending payload num 485 of size 494
 Master : Linux Kernal Space : Received payload num 485 of size 494
 Master : Linux Kernal Space : Sending payload num 486 of size 495
 Master : Linux Kernal Space : Received payload num 486 of size 495
 Master : Linux Kernal Space : Sending payload num 487 of size 496
 Master : Linux Kernal Space : Received payload num 487 of size 496
 ******************************************
 Echo Test Results: Error count = 0
 ******************************************
root@Xilinx-ZC702-AMP-14_7:~# virtio_rpmsg_bus virtio0: destroying
channel rpmsg-openamp-demo-channel addr 0x1
```

You can shut downthe firmware using the following set of commands:

```
root@Xilinx-ZC702-2013_3:~# modprobe -r rpmsg_echo_test_kern_app
root@Xilinx-ZC702-2013_3:~# modprobe -r virtio_rpmsg_bus
root@Xilinx-ZC702-2013_3:~# modprobe -r zynq_remoteproc_driver
```

2. Execute the User Space Echo Application.

   a. Load Zynq platform remoteproc module

      - For bare metal based remote echo firmware:

        ```
        modprobe zynq_remoteproc_driver firmware=
        ”/lib/firmware/zc702evk/baremetal/echo_test/firmware”
        ```

b. Load the rpmsg user device driver using the following command:

```
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_user_dev_driver
```

c. Start echo_test user-space application

```
root@Xilinx-ZC702-2013_3:~# echo_test
```

The echo test user-space application output should look similar to:

```
root@Xilinx-ZC702-AMP-2013_3:/# modprobe rpmsg_user_dev_driver
rpmsg_user_dev_driver rpmsg0: new channel: 0x400 -> 0x1!
root@Xilinx-ZC702-AMP-2013_3:/# echo_test
 Echo test start
 Open rpmsg dev!
 Query internal info ..
 rpmsg kernel fifo size = 2048
rpmsg kernel fifo free space = 2048
 ****************************************
 Please enter command and press enter key
 ****************************************
 1 - Send data to remote core, retrieve the echo and validate its
integrity ..
 2 - Quit this application ..
 CMD>1
 sending payload number 0 of size 9
 received payload number 0 of size 9
 sending payload number 2 of size 10
 received payload number 2 of size 10
 sending payload number 3 of size 11
 received payload number 3 of size 11
 sending payload number 4 of size 12
 received payload number 4 of size 12
 sending payload number 5 of size 13
 received payload number 5 of size 13
 sending payload number 6 of size 14
 received payload number 6 of size 14
 ...
 ...
 ...

 sending payload number 478 of size 486
 received payload number 478 of size 486
 sending payload number 479 of size 487
 received payload number 479 of size 487
 sending payload number 480 of size 488
 received payload number 480 of size 488
 sending payload number 481 of size 489
 received payload number 481 of size 489
 sending payload number 482 of size 490
 received payload number 482 of size 490
 sending payload number 483 of size 491
 received payload number 483 of size 491
 sending payload number 484 of size 492
 received payload number 484 of size 492
 sending payload number 485 of size 493
 received payload number 485 of size 493
 sending payload number 486 of size 494
 received payload number 486 of size 494
 sending payload number 487 of size 495
 received payload number 487 of size 495
 **************************************
 Test Results: Error count = 0
 **************************************
 ****************************************
 Please enter command and press enter key
 ****************************************
 1 - Send data to remote core, retrieve the echo and validate its
integrity ..
 2 - Quit this application ..
 CMD>
```

3. You can shutdown the firmware by using the following set of commands:

```
root@Xilinx-ZC702-2013_3:~# modprobe -r rpmsg_user_dev_driver
root@Xilinx-ZC702-2013_3:~# modprobe -r virtio_rpmsg_bus
root@Xilinx-ZC702-2013_3:~# modprobe -r zynq_remoteproc_driver
```

Related Topics

Executing Applications and Tests for Linux Master/ Baremetal Remote Configurations

Executing the Proxy Application

Executing the Proxy Application

This procedure describes the usage of the Linux user space proxy application, which provides an interface to the OpenAMP Framework proxy infrastructure.

Procedure

1. The proxy application help can be accessed as shown below:

   - to display proxy application help, execute:

     ```
     proxy_app -h
     ```

   Output:

   ```
   Linux proxy application.
   -r     Displays proxy application version.
   -f     Accepts path of firmware to load on remote core.
   -h     Displays this help message.
   ```

2. In order to bring up RPC demo firmware on the remote core, execute:

   - for bare metal remote:

   ```
   proxy_app -f /lib/firmware/zc702evk/baremetal/rpc_demo/firmware
   ```

Note

Once you have selected 'no' for repeat demo option and application quits, you are required to enter CTRL+C to quit the proxy application.

The RPC demo output should look similar to:

```
root@Xilinx-ZC702-AMP-14_7:~# proxy_app -f
/lib/firmware/zc702evk/baremetal/rpc_demo/firmware

Master>Loading remote firmware
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.

Master>Create rpmsg proxy device

Master>Opening rpmsg proxy device
 remoteproc0: powering up 0.remoteproc-test
 remoteproc0: Booting fw image firmware, size 263444
 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-openamp-demo-channel addr
0x1
rpmsg_proxy_dev_driver rpmsg0: new channel: 0x400 -> 0x1!

Master>RPC service started !!

Remote>******************************************************

Remote>Baremetal Remote Procedure Call (RPC) Demonstration

Remote>******************************************************

Remote>Rpmsg based retargetting to proxy initialized..

Remote>FileIO demo using open, write, read, close CRTL calls ..

Remote>Creating a file on master and writing to it..

Remote>Opened file 'remote.file' with fd = 4

Remote>Wrote to fd = 4, size = 45 bytes
content = This is a test string being written to file..

Remote>Closed fd = 4

Remote>Reading a file on master and displaying its contents..

Remote>Opened file 'remote.file' with fd = 4

Remote>Read from fd = 4, size = 45 bytes
content = This is a test string being written to file..
```

```
Remote>Closed fd = 4

Remote>Remote firmware using scanf and printf ..

Remote>Scanning user input from master..

Remote>Enter name
bob

Remote>Enter age
33

Remote>Enter value for pi
3.14

Remote>User name = 'bob'

Remote>User age = '33'

Remote>User entered value of pi = '3.140000'

Remote>Repeat demo ? (enter yes or no)
yes

Remote>Remote firmware using scanf and printf ..

Remote>Scanning user input from master..

Remote>Enter name
zack

Remote>Enter age
33

Remote>Enter value for pi
3.14

Remote>User name = 'zack'

Remote>User age = '33'

Remote>User entered value of pi = '3.140000'

Remote>Repeat demo ? (enter yes or no)
no

Remote>RPC retargetting quitting ...

Remote> Firmware's rpmsg-openamp-demovirtio_rpmsg_bus virtio0: destroying
channel rpmsg-openamp-demo-channel addr 0x1
-channel going down!

Master>RPC service exiting !!
 remoteproc0: stopped remote processor 0.remoteproc-test
zynq_remoteproc 0.remoteproc-test: zynq_remoteproc_remove
zynq_remoteproc 0.remoteproc-test: Deleting the irq_list
 remoteproc0: releasing 0.remoteproc-test
```

```
CPU1: Booted secondary processor
```

3. Verify the file created by remote context.

   ```
   root@Xilinx-ZC702-AMP-14_7:~# cat remote.file
   ```

   This is a test string being written to file.

Related Topics

Executing Applications and Tests for Linux Master/ Baremetal Remote Configurations

Executing the Matrix Multiply Application

Build and Execute Applications for Baremetal Master/ Linux Remote Configurations

The following steps will generate a Linux remote kernel image using PetaLinux workflow. Then an OpenAMP Framework build will generate bare metal master applications which will be able to boot this Linux image.

Building a Linux Remote Kernel Image

The following steps generate a Linux kernel image, using the PetaLinux workflow. Then an OpenAMP Framework build will generate bare metal master applications which will be able to boot this Linux image.

Note

The provided procedure will patch the PetaLinux source code for a unicore configuration. If you want to retain an original copy of the Linux kernel source, it is recommended that you perform another installation of the PetaLinux for remote kernel, go through the environment setup and build the remote kernel image from this new installation.

Procedure

1. Patch Linux Source

   - Patch the gic handler within PetaLinux kernel source tree as:

     ```
     $ patch $PETALINUX/components/linux-kernel/xlnx-
     3.8/arch/arm/common/gic.c <
     $OPENAMP/libs/system/zc702evk/linux/patches/linux/petalinux2013.10/
           gic.patch
     ```

   - Patch the devtree component as:

     ```
     $patch $PETALINUX/components/linux-kernel/xlnx-
     3.8/arch/arm/kernel/devtree.c  <
     $OPENAMP/libs/system/zc702evk/linux/patches/linux/petalinux2013.10/
     devtree.patch
     ```

Note

This patch modifies the Linux kernel source to enable it to co-operatively operate in a remote context with bare metal based master.

Once this patch is applied, in order to rebuild the SMP Linux kernel project for master context, this patch should be removed.

2. Create a project for the remote Linux unicore kernel:

   Change directory to the location where remote Linux unicore kernel project needs to be created and execute the following:

   ```
        >petalinux-create -t project -n <remote_name> --template zynq
   ```

   where <*remote_name*> is the user defined name of the remote Linux unicore kernel project.

Note

This command creates a project within a folder named <*remote name*> under the present working directory. Here forth, <*remote_root*> will refer to this location of remote unicore project.

The structure of this project should be similar to:

```
<remote_root>
      "hw-description"
      "subsystems"
         "linux"
            "hw-decsription"
               Systems.dts
      "config.project"
```

3. Create Linux Kernel-Space and User-Space Applications

Assuming that the remote project has been created successfully and $OPENAMP is set to the root of OpenAMP Framework source tree, execute the following command:

```
$ source $OPENAMP/libs/system/zc702evk/linux/scripts/
      open_amp_create_projects.sh remote <remote_root>
```

where <*remote_root*> is the path of the remote Linux project.

It will generate the following user space application and modules:

```
1. <remote_root>/components/apps/mat_mul_demo
2. <remote_root>/components/apps/echo_test
3. <remote_root>/components/modules/zynq_rpmsg_driver
4. <remote_root>/components/modules/rpmsg_mat_mul_kern_app
5. <remote_root>/components/modules/rpmsg_echo_test_kern_app
6. <remote_root>/components/modules/rpmsg_user_dev_driver
7. <remote_root>/components/modules/rpmsg_func_test_kern_app
```

4. Configure the System Memory Map

    a. Setup PetaLinux so that the Linux kernel address starts at address 0x00000000. Execute:

    ```
    $ cd <remote_root>
    $ petalinux-config
    ```

    b. Change the Kernel base address value to 0x00000000 if it is not already set to this value.

    ```
    *** linux Components Selection ***
    ...
    (0x00000000) Kernel base address
    ```

5. Configure the PetaLinux kernel

    Ensure the following kernel configurations:

    a. Execute the following command while in the <*remote_root*> directory

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

    b. Select Enable loadable module support.

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

    c. Disable Symmetric Multi-Processing within Kernel Features

    ```
    Kernel Configuration --->
         Kernel Features --->
              [ ] Symmetric Multi-Processing
    ```

    d. Select High Memory Support within Kernel Features

    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
```

e. 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
```

6. Build PetaLinux

```
$ cd <remote_root>

$ petalinux-build
```

7. Copy the PetaLinux image to OPENAMP

   Copy the PetaLinux image:

```
<remote_root>/images/linux/image.ub
```

   to the following OPENAMP directory

```
$OPENAMP/libs/system/zc702evk/linux
```

   using the following command

```
$ cp <remote_root>/images/linux/image.ub
$OPENAMP/libs/system/zc702evk/linux/image.ub
```

Building the OpenAMP Framework Library, Applications and Tests

An OpenAMP Framework build generates bare metal master applications which can boot the Linux image.

Prerequisites

- The OpenAMP Framework library is installed at location $OPENAMP.

Procedure

1. Change directory to $OPENAMP

```
$ cd $OPENAMP
```

2. Execute the build script. This operation builds remote applications for the bare metal environment.

Note
The build operation generates application outputs in the respective application directories.

```
$ source open_amp_build.sh
```

Related Topics

Building a Linux Remote Kernel Image

Executing Applications and Tests for Baremetal Master/Linux Remote Configuration

This procedure shows the steps for moving U-Boot, application, and test binaries to boot medium (SD Card).

Prerequisites

- The U-Boot binary image (*BOOT.bin*), and OpenAMP Framework application and test binaries are built and available.

Procedure

1. Format an SDCARD and partition for FAT.
2. Insert the SDCARD in the host machine and place the generated U-Boot image (*BOOT.bin*) for ZC702EVK, OpenAMP Framework binary images available at locations listed below should be moved to SDCARD as well.
   - *apps/samples/master/baremetal/matrix_multiply/matrix_multiply.bin*
   - *apps/tests/master/baremetal/echo_test/echo_test.bin*
   - *apps/tests/master/baremetal/func_test_suite/func_test_suite.bin*
3. Insert the SDCARD in the SDIO slot J64 on ZC702EVK.
4. Power up the board to boot U-Boot
5. Once U-Boot is up, in the U-Boot console, enter

```
U-Boot-PetaLinux> dcache off;
fatload mmc <dev>:<part> 0x10000000 <baremetal app>.bin;
go 0x10000000
```

Note
<dev> refers to the relevant MMC device number. The list of available devices can be retrieved from the U-Boot prompt by entering - U-Boot-PetaLinux> mmc list.
<part> refers to the relevant partition on the current mmc device. It can be retrieved from the U-Boot prompt by entering - U-Boot-PetaLinux> mmc part.

Note
The following sections assume that the relevant <dev>:<part> value is 0:0. The actual values may vary and you are advised to confirm the device and partition numbers first.

Related Topics

Executing Applications and Tests for Baremetal Master/Linux Remote Configuration

Executing the Matrix Multiply Application for Baremetal Master/Linux Remote

Executing the Matrix Multiply Application for Baremetal Master/Linux Remote

The following procedure outlines the instructions to execute the matrix multiplication demo application. You can execute one of the two Linux remote applications (kernelspace or userspace). In order to execute the other application, you need to perform a power reset and restart the bare metal master matrix_multiply application.

Procedure

1. Start the matrix multiply application from the U-Boot prompt:

```
U-Boot-PetaLinux> dcache off;
fatload mmc 0:0 0x10000000 matrix_multiply.bin;
go 0x10000000
```

   It starts the bare metal application on the primary core which will on its own boot the Linux kernel on remote core. The Linux kernel will boot and capture the console.

2. Once the Linux kernel prompt is up, enter root login and password:

```
Xilinx-ZC702-2013_3 login: root
Password:
login[776]: root login  on `ttyPS0'
```

3. Load Zynq rpmsg driver module

```
root@Xilinx-ZC702-2013_3:~# modprobe zynq_rpmsg_driver
```

   - User Space

     i. Load the rpmsg user device driver using the following command:

```
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_user_dev_driver
```

ii. Start matrix_multiply user-space application

```
root@Xilinx-ZC702-2013_3:~# mat_mul_demo
```

- Kernel Space
  - Load the mat mul apps kernel driver using the following command:

```
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_mat_mul_kern_app
```

Results

User space

```
## Starting application at 0x10000000 ...

*********************************
OpenAMP Linux Bootstrap.
*********************************

Linux Bootstrap: Locating Linux Kernel and DTB from FIT image.

Linux Bootstrap: Kernel image is compressed. Starting decompression
process. It may take a while...

Linux Bootstrap: Linux image decompression complete.
Linux Bootstrap: Linux kernel image has been loaded into memory.
Linux Bootstrap: Loaded DTB.
Linux Bootstrap: Booting Linux.
Booting Linux on physical CPU 0x0
Linux version 3.8.11 (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-
40) ) #99 PREEMPT Fri May 2 15:40:46 PKT 2014
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Xilinx Zynq Platform, model: .
Memory policy: ECC disabled, Data cache writeback
…
…
starting Busybox inet Daemon: inetd... done.
Starting uWeb server:
INIT: Entering runlevel: 5
Stopping Bootlog daemon: bootlogd.

 _____        _         _     _
| ___ \      | |       | |   (_)
| |_/ / ___ | |_  __ _ | |    _  _ __   _   _ __  __
|  __/ / _ \| __| / _` || |   | || '_ \ | | | |\ \/ /
| |    |  __/| |_ | (_| || |____| || | | || |_| | >  <
\_|     \___| \__| \__,_|\_____/|_||_| |_| \__,_|/_/\_\

PetaLinux v2013.10 (Yocto 1.4) Xilinx-ZC702-2013_3 ttyPS0

Xilinx-ZC702-2013_3 login: root
Password:
login[776]: root login  on `ttyPS0'
```

```
root@Xilinx-ZC702-2013_3:~# modprobe zynq_rpmsg_driver
zynq_rpmsg_driver 8000000.zynq-rpmsg_driver: virtio device registered
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_user_dev_driver
virtio_rpmsg_bus virtio0: rpmsg host is online
virtio_rpmsg_bus virtio0: creating channel rpmsg-openamp-demo-channel addr
0x1
rpmsg_user_dev_driver rpmsg0: new channel: 0x400 -> 0x1!
root@Xilinx-ZC702-2013_3:~# mat_mul_demo

 Matrix multiplication demo start

 Open rpmsg dev!

 Query internal info ..
 rpmsg kernel fifo size = 2048
 rpmsg kernel fifo free space = 2048

 Creating ui_thread and compute_thread ...

 ****************************************
 Please enter command and press enter key
 ****************************************
 1 - Generates random 6x6 matrices and transmits them to remote core over
rpmsg ..
 2 - Quit this application ..
 CMD>1

 Compute thread unblocked ..
 The compute thread is now blocking on a read() from rpmsg device

 Generating random matrices now ...

 Master : Linux : Input matrix 0

 0  5  0  0  1  6
 8  0  1  6  5  2
 7  0  1  2  2  2
 3  7  7  1  3  7
 4  5  9  2  8  9
 3  1  4  5  3  8

 Master : Linux : Input matrix 1

 1  1  8  2  8  4
 6  7  6  7  2  8
 2  7  7  9  8  2
 8  2  8  9  7  8
 0  2  9  7  7  5
 5  0  6  3  2  6

 Writing generated matrices to rpmsg device, 296 bytes written ..

 Received results! - 148 bytes from rpmsg device (transmitted from remote
context)

 Master : Linux : Printing results
 60  37  75  60  29  81
 68  37  176  120  153  119
 35  22  109  61  96  68
 102  109  192  169  136  147
 113  122  267  225  202  184
 97  54  173  139  130  131

 ****************************************
 Please enter command and press enter key
 ****************************************
 1 - Generates random 6x6 matrices and transmits them to remote core over
rpmsg ..
 2 - Quit this application ..
 CMD>1

 Compute thread unblocked ..
 The compute thread is now blocking on a read() from rpmsg device

 Generating random matrices now ...

 Master : Linux : Input matrix 0

 0  0  2  1  5  7
 0  5  2  4  2  3
 8  5  5  6  5  3
 9  1  5  0  3  5
 3  2  5  7  2  4
 7  4  5  1  5  2

 Master : Linux : Input matrix 1

 8  8  7  2  2  0
 5  3  7  2  9  2
 5  0  4  1  0  7
 6  5  0  3  2  4
 0  9  8  5  0  5
 9  0  3  6  2  6

 Writing generated matrices to rpmsg device, 296 bytes written ..

 Received results! - 148 bytes from rpmsg device (transmitted from remote
context)

 Master : Linux : Printing results
 79  50  69  72  16  85
 86  53  68  52  59  68
 177  154  160  92  79  112
 147  102  129  70  37  82
 137  83  83  70  46  101
 125  118  143  67  56  84

 ****************************************
 Please enter command and press enter key
 ****************************************
 1 - Generates random 6x6 matrices and transmits them to remote core over
rpmsg ..
 2 - Quit this application ..
 CMD>2
 Quitting application ..
 Matrix multiplication demo end
INIT: Sending processes the TERM signal

Broadcast message from root@Xilinx-ZC702-2013_3 (Thu Jan  1 00:01:03
1970):
The system is going down for system halt NOW!
INIT: Sending processes the KILL signal
Deconfiguring network interfaces... done.
Sending all processes the TERM signal...
Sending all processes the KILL signal...
Unmounting remote filesystems...
Deactivating swap...
Unmounting local filesystems...
System halted.
```

Kernel space

```
*********************************
OpenAMP Linux Bootstrap.
*********************************

Linux Bootstrap: Locating Linux Kernel and DTB from FIT image.
Linux Bootstrap: Kernel image is compressed. Starting decompression
process. It may take a while...
Linux Bootstrap: Linux image decompression complete.
Linux Bootstrap: Linux kernel image has been loaded into memory.
Linux Bootstrap: Loaded DTB.
Linux Bootstrap: Booting Linux.

Booting Linux on physical CPU 0x0
Linux version 3.8.11 (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-
40) ) #4 PREEMPT Wed May 7 18:46:05 PKT 2014
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Xilinx Zynq Platform, model: .
Memory policy: ECC disabled, Data cache writeback
...
...
...
done.
starting Busybox inet Daemon: inetd... done.
Starting uWeb server:
INIT: Entering runlevel: 5
Stopping Bootlog daemon: bootlogd.

 _____      _          _     _
| ___ \    | |        | |    (_)
| |_/ / ___ | |_  __ _ | |     _  _ __  _   _ __  __
|  __/ / _ \| __| / _` || |    | || '_ \ | | | |\ \/ /
| |   |  __/| |_ | (_| || |____| || | | || |_| | >  <
\_|    \___| \__| \__,_|\_____/|_||_| |_| \__,_|/_/\_\

PetaLinux v2013.10 (Yocto 1.4) Xilinx-ZC702-2013_3 ttyPS0

Xilinx-ZC702-2013_3 login: root
Password:
```

```
login[776]: root login  on `ttyPS0'

root@Xilinx-ZC702-2013_3:~# modprobe zynq_rpmsg_driver
zynq_rpmsg_driver 8000000.zynq-rpmsg_driver: virtio device registered
root@Xilinx-ZC702-2013_3:~#
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_mat_mul_kern_app
virtio_rpmsg_bus virtio0: rpmsg host is online
virtio_rpmsg_bus virtio0: creating channel rpmsg-openamp-demo-channel addr
0x1
rpmsg_mat_mul_kern_app: module license 'unspecified' taints kernel.
Disabling lock debugging due to kernel taint

 Demo Start - Demo rpmsg driver got probed
 since the rpmsg device associated with driver was found !

 Create endpoint and register rx callback

 Master : Linux : Generating random matrices

 Master : Linux : Input matrix 0

 5  0  9  3  1  5
 0  9  6  1  9  0
 2  2  0  4  4  5
 6  2  5  6  3  2
 3  5  1  1  2  4
 7  1  3  3  2  7

 Master : Linux : Input matrix 1

 3  7  7  6  7  9
 9  1  4  4  5  5
 4  5  4  3  6  1
 7  0  9  2  2  7
 5  8  5  0  7  2
 1  9  3  1  9  2

 Master : Linux : Sent 296 bytes of data over rpmsg channel to remote

 Master : Linux : Received 148 bytes of data over rpmsg channel from
remote
root@Xilinx-ZC702-2013_3:~#
 Master : Linux : Printing results
 82  133  118  68  147  87
 157  111  114  56  146  76
 77  93  93  33  105  74
 115  111  145  73  133  121
 79  83  76  47  104  72
 80  144  123  68  155  110

 Master : Linux : Received 4 bytes of data over rpmsg channel from remote
INIT: Sending processes the TERM signal

Broadcast message from root@Xilinx-ZC702-2013_3 (Thu Jan  1 00:02:41
1970):
The system is going down for system halt NOW!
INIT: Sending processes the KILL signal
Deconfiguring network interfaces... done.
```

```
Sending all processes the TERM signal...
Sending all processes the KILL signal...
Unmounting remote filesystems...
Deactivating swap...
Unmounting local filesystems...
System halted.
```

Related Topics

Executing Applications and Tests for Baremetal Master/Linux Remote Configuration

Executing the Echo Test Application

Executing the Echo Test Application

The following procedure outlines the instructions to execute the echo test application. You can execute one of the two Linux remote applications (kernelspace or userspace). In order to execute the other application,you need to perform a power reset and restart the bare metal master echo test application.

Procedure

1. Start the echo test application from the U-Boot prompt:

   ```
   U-Boot-PetaLinux> dcache off;fatload mmc 0:0 0x10000000
   echo_test.bin; go 0x10000000
   ```

   It starts the bare metal application on the primary core which will on its own boot the Linux kernel on remote core. The Linux kernel will boot and capture the console.

2. Once the Linux kernel prompt is up, enter root login and password:

   ```
   Xilinx-ZC702-2013_3 login: root
   Password:
   login[776]: root login  on `ttyPS0'
   ```

3. Load Zynq rpmsg driver module

   ```
   root@Xilinx-ZC702-2013_3:~# modprobe zynq_rpmsg_driver
   ```

   - User Space

     i. Load the rpmsg user device driver using the following command:

     ```
     root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_user_dev_driver
     ```

     ii. Start echo_test user-space application

     ```
     root@Xilinx-ZC702-2013_3:~# echo_test
     ```

   - Kernel Space

     - Load the echo test driver using the following commands:

```
root@Xilinx-ZC702-2013_3:~# modprobe virtio_rpmsg_bus
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_echo_test_kern_app
```

Results

User space

```
## Starting application at 0x10000000 ...

*********************************
OpenAMP Linux Bootstrap.
*********************************

Linux Bootstrap: Locating Linux Kernel and DTB from FIT image.

Linux Bootstrap: Kernel image is compressed. Starting decompression
process. It may take a while...

Linux Bootstrap: Linux image decompression complete.

Linux Bootstrap: Linux kernel image has been loaded into memory.

Linux Bootstrap: Loaded DTB.

Linux Bootstrap: Booting Linux.
Booting Linux on physical CPU 0x0
Linux version 3.8.11 (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-
40) ) #99 PREEMPT Fri May 2 15:40:46 PKT 2014
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Xilinx Zynq Platform, model: .
Memory policy: ECC disabled, Data cache writeback
...
...
...

/etc/udhcpc.d/50default: Adding DNS 137.202.187.16
/etc/udhcpc.d/50default: Adding DNS 137.202.23.16
/etc/udhcpc.d/50default: Adding DNS 147.34.2.16
done.
starting Busybox inet Daemon: inetd... done.
Starting uWeb server:
INIT: Entering runlevel: 5
Stopping Bootlog daemon: bootlogd.

 _____      _          _      _
| ___ \     | |        | |     (_)
| |_/ / ___ | |_   __ _ | |     _  _ __   _   _ __  __
|  __/ / _ \| __| / _` || |    | || '_ \ | | | |\ \/ /
| |    |  __/| |_ | (_| || |____| || | | || |_| | >  <
\_|     \___| \__| \__,_|\_____/|_||_| |_| \__,_|/_/\_\

PetaLinux v2013.10 (Yocto 1.4) Xilinx-ZC702-2013_3 ttyPS0

Xilinx-ZC702-2013_3 login: root
Password:
login[776]: root login  on `ttyPS0'
```

```
root@Xilinx-ZC702-2013_3:~# modprobe zynq_rpmsg_driver
zynq_rpmsg_driver 8000000.zynq-rpmsg_driver: virtio device registered
root@Xilinx-ZC702-2013_3:~#
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_user_dev_driver
virtio_rpmsg_bus virtio0: rpmsg host is online
virtio_rpmsg_bus virtio0: creating channel rpmsg-openamp-demo-channel addr
0x1
rpmsg_user_dev_driver rpmsg0: new channel: 0x400 -> 0x1!
root@Xilinx-ZC702-2013_3:~# echo_test

 Echo test start

 Open rpmsg dev!

 Query internal info ..
 rpmsg kernel fifo size = 2048
 rpmsg kernel fifo avail data = 4096
 rpmsg kernel fifo free space = 2048

 ****************************************
 Please enter command and press enter key
 ****************************************
 1 - Send data to remote core, retrieve the echo and validate its
integrity ..
 2 - Quit this application ..
 CMD>1

 sending payload number 0 of size 9
 received payload number 0 of size 9

 sending payload number 2 of size 10
 received payload number 2 of size 10

 sending payload number 3 of size 11
 received payload number 3 of size 11

 ...
 ...
 ...

 sending payload number 485 of size 493
 received payload number 485 of size 493

 sending payload number 486 of size 494
 received payload number 486 of size 494

 sending payload number 487 of size 495
 received payload number 487 of size 495

 **************************************

 Test Results: Error count = 0

 **************************************

 ****************************************
 Please enter command and press enter key
 ****************************************
 1 - Send data to remote core, retrieve the echo and validate its
integrity ..
 2 - Quit this application ..
 CMD>2
INIT: Sending processes the TERM signal

Broadcast message from root@Xilinx-ZC702-2013_3 (Thu Jan  1 00:01:23
1970):
The system is going down for system halt NOW!
INIT: Sending processes the KILL signal
Deconfiguring network interfaces... done.
Sending all processes the TERM signal...
Sending all processes the KILL signal...
Unmounting remote filesystems...
Deactivating swap...
Unmounting local filesystems...
System halted.
```

Kernel space

```
*********************************
OpenAMP Linux Bootstrap.
*********************************
Linux Bootstrap: Locating Linux Kernel and DTB from FIT image.
Linux Bootstrap: Kernel image is compressed. Starting decompression
process. It may take a while...
Linux Bootstrap: Linux image decompression complete.
Linux Bootstrap: Linux kernel image has been loaded into memory.
Linux Bootstrap: Loaded DTB.
Linux Bootstrap: Booting Linux.

Booting Linux on physical CPU 0x0
Linux version 3.8.11 (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-
40) ) #4 PREEMPT Wed May 7 18:46:05 PKT 2014
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Xilinx Zynq Platform, model: .
Memory policy: ECC disabled, Data cache writeback
...
...
...
starting Busybox inet Daemon: inetd... done.
Starting uWeb server:
INIT: Entering runlevel: 5
Stopping Bootlog daemon: bootlogd.

 _____       _        _     _
| ___ \     | |      | |   (_)
| |_/ / ___ | |_  __ _| |    _ _ __  _   ___  __
|  __/ / _ \| __| / _` || |   | || '_ \ | | | |\ \/ /
| |   |  __/| |_ | (_| || |____| || | | || |_| | >  <
\_|    \___| \__| \__,_|\_____/|_||_| |_| \__,_|/_/\_\

PetaLinux v2013.10 (Yocto 1.4) Xilinx-ZC702-2013_3 ttyPS0

Xilinx-ZC702-2013_3 login: root
```

```
Password:
login[776]: root login  on `ttyPS0'

root@Xilinx-ZC702-2013_3:~# modprobe zynq_rpmsg_driver
zynq_rpmsg_driver 8000000.zynq-rpmsg_driver: virtio device registered
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_echo_test_kern_app
virtio_rpmsg_bus virtio0: rpmsg host is online
virtio_rpmsg_bus virtio0: creating channel rpmsg-openamp-demo-channel addr
0x1
rpmsg_echo_test_kern_app: module license 'unspecified' taints kernel.
Disabling lock debugging due to kernel taint

 Echo Test Start!

 Master : Linux Kernal Space : Sending payload num 0 of size 9

 Master : Linux Kernal Space : Received payload num 0 of size 9

 Master : Linux Kernal Space : Sending payload num 1 of size 10

 Master : Linux Kernal Space : Received payload num 1 of size 10
 ...
 ...
 ...
 Master : Linux Kernal Space : Sending payload num 486 of size 495

 Master : Linux Kernal Space : Received payload num 486 of size 495

 Master : Linux Kernal Space : Sending payload num 487 of size 496

 Master : Linux Kernal Space : Received payload num 487 of size 496

 *******************************************

 Echo Test Results: Error count = 0

 *******************************************

 Master : Linux Kernal Space : Received payload num -279534246 of size 4
root@Xilinx-ZC702-2013_3:~#
Broadcast message from root@Xilin +Ëëdown for system halt NOW!
INIT: Sending processes the TERM signal
INIT: Sending processes the KILL signal
Deconfiguring network interfaces... done.
Sending all processes the TERM signal...
Sending all processes the KILL signal...
Unmounting remote filesystems...
Deactivating swap...
Unmounting local filesystems...
```

Related Topics

Executing Applications and Tests for Baremetal Master/Linux Remote Configuration

Executing the Functional Test Suites

Executing the Functional Test Suites

The following procedure outlines the instructions to execute the functional test suite application.

Note

The target needs to be power cycled after running this application in order to run another application.

Procedure

1. Start the functional test suites application from the U-Boot prompt:

   ```
   U-Boot-PetaLinux> dcache off;fatload mmc 0:0 0x10000000
   func_test_suite.bin; go 0x10000000
   ```

   It starts the bare metal application on the primary core which will on its own boot the Linux kernel on remote core. The Linux kernel will boot and capture the console.

2. Once the Linux kernel prompt is up, enter root login and password:

   ```
   Xilinx-ZC702-2013_3 login: root
   Password:
   login[776]: root login  on `ttyPS0'
   ```

3. Load the Zynq rpmsg driver module

   ```
   root@Xilinx-ZC702-2013_3:~# modprobe zynq_rpmsg_driver
   ```

4. Load the functional test suite driver using the following command:

   ```
   root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_func_test_kern_app
   ```

   Once the test suite is completed, Linux will shutdown.

Results

```
## Starting application at 0x10000000 ...

*********************************
OpenAMP Linux Bootstrap.
*********************************

Linux Bootstrap: Locating Linux Kernel and DTB from FIT image.
Linux Bootstrap: Kernel image is compressed. Starting decompression
process. It may take a while...
Linux Bootstrap: Linux image decompression complete.
Linux Bootstrap: Linux kernel image has been loaded into memory.
Linux Bootstrap: Loaded DTB.

Linux Bootstrap: Booting Linux.
Booting Linux on physical CPU 0x0
Linux version 3.8.11 (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-
40) ) #99 PREEMPT Fri May 2 15:40:46 PKT 2014
```

```
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Xilinx Zynq Platform, model: .
Memory policy: ECC disabled, Data cache writeback
...
...
starting Busybox inet Daemon: inetd... done.
Starting uWeb server:
INIT: Entering runlevel: 5
Stopping Bootlog daemon: bootlogd.

 _____        _          _      _
| ___ \      | |        | |    (_)
| |_/ / ___ | |_   __ _ | |     _  _ __   _   _ __  __
|  __/ / _ \| __| / _` || |    | || '_ \ | | | |\ \/ /
| |   |  __/| |_ | (_| || |____| || | | || |_| | >  <
\_|    \___| \__| \__,_|\______/|_||_| |_| \__,_|/_/\_\

PetaLinux v2013.10 (Yocto 1.4) Xilinx-ZC702-2013_3 ttyPS0

Xilinx-ZC702-2013_3 login: root
Password:
login[776]: root login  on `ttyPS0'

root@Xilinx-ZC702-2013_3:~# modprobe zynq_rpmsg_driver
zynq_rpmsg_driver 8000000.zynq-rpmsg_driver: virtio device registered
root@Xilinx-ZC702-2013_3:~# modprobe rpmsg_func_test_kern_app
virtio_rpmsg_bus virtio0: rpmsg host is online
virtio_rpmsg_bus virtio0: creating channel rpmsg-openamp-demo-channel addr
0x1
rpmsg_func_test_kern_app: module license 'unspecified' taints kernel.
Disabling lock debugging due to kernel taint

Func Test Suite Start!

RPMSG Send Test: Passed

RPMSG Send Offchannel Test: Passed

RPMSG Create EPT Test: Passed

Channel Deletion. Shutdown would be next
root@Xilinx-ZC702-2013_3:~#
Broadcast message from root@Xilinoing down for system halt NOW!
INIT: Sending processes the TERM signal
INIT: Sending processes the KILL signal
Deconfiguring network interfaces... done.
Sending all processes the TERM signal...
Sending all processes the KILL signal...
Unmounting remote filesystems...
Deactivating swap...
Unmounting local filesystems...
System halted.
```

Related Topics

Executing Applications and Tests for Baremetal Master/Linux Remote Configuration

Executing the Matrix Multiply Application for Baremetal Master/Linux Remote

Third-Party Information

Mentor Graphics BSD License, v1.0