BORA

ARM Cortex-A9 + FPGA CPU Module

Ultra Line

Bora Embedded Linux Kit (*BELK*)

AN-BELK-001

Asymmetric Multiprocessing (AMP) on Bora – Linux + FreeRTOS

<Page intentionally left blank>

Table of Contents

1 Preface

1.1 About this document

This application note describes how to run a simple application on FreeRTOS porting for Zynq, running on Zynq core #2.

1.2 Copyrights/Trademarks

1.3 Standards

DAVE Embedded Systems is certified to ISO 9001 standards.

1.4 Disclaimers

DAVE Embedded Systems does not assume any responsibility for availability, supply and support related to all products mentioned in this document that are not strictly part of the Bora CPU module and the BoraEVB carrier board.

Bora CPU Modules are not designed for use in life support appliances, devices, or systems where malfunctioning of these products can reasonably be expected to result in personal injury. **DAVE Embedded Systems** customers who are using or selling these products for use in such applications do so at their own risk and agree to fully indemnify **DAVE Embedded Systems** for any damage resulting from such improper use or sale.

1.5 Technical Support

We are committed to making our products easy to use and will help customers use our CPU modules in their systems.

Technical support is delivered through email for registered kits owners. Support requests can be sent to support-bora@dave.eu. Software upgrades are available for download in the restricted download area of **DAVE Embedded Systems** web site: http://www.dave.eu/reserved-area. An account is required to access this area.

Please refer to our Web site at http://www.dave.eu/products/zynq-bora for the latest product documents, utilities, drivers, Product Change Notices, Board Support Packages, Application Notes, mechanical drawings and additional tools and software.

1.6 Related documents

Document	Location
DAVE Embedded Systems Developers Wiki	http://wiki.dave.eu/index.php/Main_Page
Zynq-7000 Technical Reference Manual	http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf
Bora main page on **DAVE Embedded Systems** Developers Wiki	http://wiki.dave.eu/index.php/Category:Bora
Bora Hardware Manual	http://www.dave.eu/sites/default/files/files/bora-hm.pdf
BoraEVB page on **DAVE Embedded Systems** Developers Wiki	http://wiki.dave.eu/index.php/BoraEVB
Vivado Design Suite User Guide: Embedded Processor Hardware Design	http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_2/ug898-vivado-embedded-design.pdf
Zynq-7000 All Programmable SoC: Concepts, Tools, and Techniques (CTT)	http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_6/ug873-zynq-ctt.pdf
Zynq-7000 All Programmable SoC Software Developers Guide	http://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf
BELK Quick Start Guide	Provided with BELK
Xilinx UG978 (v2013.04) April 22, 2013	http://www.xilinx.com/support/documentation/sw_manuals/petalinux2013_04/ug978-petalinux-zynq-amp.pdf

Tab. 1: Related documents

1.7 Conventions, Abbreviations, Acronyms

Abbreviation	Definition
AMP	Asymmetric multiprocessing
BELK	Bora Embedded Linux Kit
FPGA	Field Programmable Gate Array
GPI	General purpose input
GPIO	General purpose input and output
GPO	General purpose output
OS	Operating System
PL	Zynq Programmable Logic
PS	Zynq Processing System
PSU	Power supply unit
SOC	System-on-chip
SOM	System-on-module
SMP	Symmetric multiprocessing

Tab. 2: Abbreviations and acronyms used in this manual

Revision History

Version	***Date***	***Notes***
1.0.0	November 2013	First release
1.0.1	November 2013	Added UART0 pinout information Minor fixes
1.1.0	November 2013	Added support for RPMsg example
1.5.0	December 2013	Added chapter related to Lauterbach debugger
1.5.1	January 2014	Minor fixes
1.6.0	April 2014	Minor fixes Updated for BELK 2.0.0 release

2 Introduction

2.1 Bora SOM

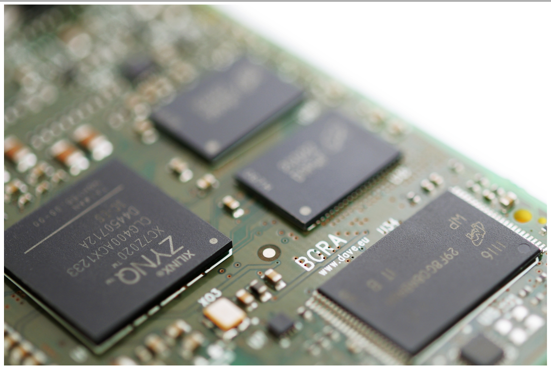


Fig. 1: Bora – Powered by Zynq processor

BORA is the new top-class Dual Cortex-A9 + FPGA CPU module by **DAVE Embedded Systems**, based on the recent Xilinx Zynq XC7Z010/XC7Z020 application processor.

Thanks to BORA, customers are going to save time and resources by using a compact solution that **includes both the CPU and the FPGA**, avoiding complexities on the carrier PCB.

The use of this processor enables extensive system-level differentiation of new applications in many industry fields, where high performances and extremely compact form factor (85mm x 50mm) are key factors. Smarter system designs are made possible, following the trends in functionalities and interfaces of the new, state-of-the-art embedded products.

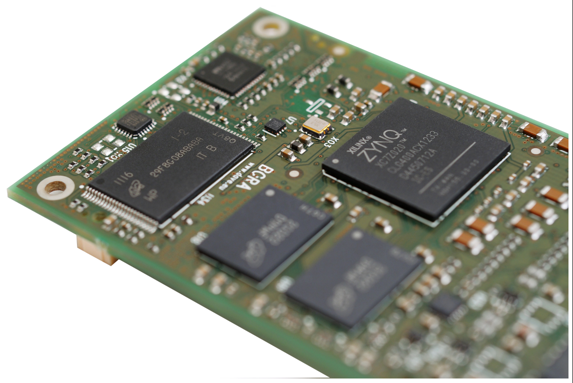


Fig. 2: Bora – Dual ARM Cortex A9 plus FPGA

BORA offers great computational power, thanks to the rich set of peripherals, the Dual Cortex-A9 and the Artix-7 FPGA together with a large set of high-speed I/Os (up to 5GHz).

BORA enables designers to create rugged products suitable for harsh mechanical and thermal environments, allowing the development of the most advanced and robust products.

Thanks to the tight integration between the ARM-based processing system and the on-chip programmable logic, designers are free to add virtually any peripheral or create custom accelerators that extend system performance and better match specific application requirements.

BORA is designed and manufactured according to **DAVE Embedded Systems *Ultra* Line** specifications, in order to guarantee premium quality and technical value for customers who require top performances and flexibility. BORA is suitable for high-end applications such as medical instrumentation, advanced communication systems, critical real-time operations and safety applications.

For further information on Bora, please refer to Bora Hardware Manual.

2.2 Xilinx Zynq 7000 SOC

The Zynq™-7000 family SOCs integrate a feature-rich dual-core ARM® Cortex™-A9 based processing system (PS) and Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 CPUs are the heart of the PS, while the PL provides a rich architecture of user-configurable capabilities. The PS and PL can be tightly or loosely coupled using multiple interfaces and other signals that have a combined total of over 3,000 connections. This enables the designer to effectively integrate user-created hardware accelerators and other functions in the PL logic that are accessible to the processors and can also access memory resources in the PS. Zynq customers are able to differentiate their product in hardware by customizing their applications using PL.

In contrast with “typical” SOCs, where developers have to deal just with one main component (the CPU), Zynq SOC adds some complexity, since both the PS part and the PL part must be managed. Therefore, some knowledge of FPGAs and how they work would be valuable. However, the design process for Zynq-based systems is PS-centric: the processors in the PS always boot first, allowing a software centric approach for PL configuration. The PL can be configured as part of the boot process or configured at some point in the future. Additionally, the PL can be completely reconfigured or used with partial, dynamic reconfiguration (PR). This latter capability is analogous to the dynamic loading and unloading of software modules. The PL configuration data is referred to as a bitstream.

2.3 Asymmetric Multiprocessing

Asymmetric Multi Processing (AMP) allows a multiprocessor system to run multiple Operating Systems (OS) that are independent of each other. In other words, each CPU has its own private memory space, which contains the OS and the applications that are to run on that CPU. In addition, there can be some shared memory space that is used for multiprocessor communication. This is contrasted with Symmetric Multiprocessing (SMP), in which one OS runs on multiple CPUs using a public shared memory space. Thanks to AMP, developers can use open-source Linux and FreeRTOS operating systems and the RPMsg Inter Processor Communication (IPC) framework between the Zynq's two high-performance ARM® Cortex™-A9 processors to quickly implement applications that need to deliver deterministic, real-time responsiveness for markets such as automotive, industrial and others with similar requirements. For further information, please refer to http://www.wiki.xilinx.com/Multi-OS+Support+%28AMP+%26+Hypervisor%29

2.4 BELK

Bora Embedded Linux Kit (BELK for short) provides all the necessary components required to set up the developing environment for:

- configuring the system (PS and PL) at hardware level
- build the first-stage bootloader (FSBL)
- building the second stage bootloader (U-Boot)
- building and running Linux operating system on Bora-based systems
- building Linux applications that will run on the target

DAVE Embedded Systems provides all the customization required (in particular at bootloader and Linux kernel levels) to enable customers use the standard Zynq-7000 development tools for building all the firmware/software components that will run on the target system.

Please refer to the **BELK Quick Start Guide** for further details on BELK.

N.B.: this application note has been tested using BELK 2.0.0.

3 AMP on Bora

The following sections describe how to build the software components required to set up asymmetric multi-processing (AMP for short) configuration required to run Linux OS on first Cortex-A9 core and FreeRTOS on second Cortex-A9 core.

Two different examples are provided. The first one – HelloWorld – shows basic functionalities while the second – RPMsg-based application – exploits more sophisticated techniques to handle inter-processors communication and synchronization. This latter configuration is based on RPMsg mechanism as described in Xilinx document **UG978** (v2013.04, April 22, 2013).

3.1 Prerequisites

- Vivado® Design Suite version **2013.3** with Xilinx SDK (Webpack license is minimum requirements)
- Python 2.7.x (C:\Python27 must be the installation directory on Windows)
- Bora Embedded Linux Kit (Please refer to BELK Quick Start Guide for further details) version **2.0.0**.
- BORA FreeRTOS repository (please refer to section 3.2.3)

3.2 Building the software components

3.2.1 Vivado project

- Start the Zynq development server and login into the system
- Assuming that a local repository has not been created, clone the remote Bora git repository (the “-b” option is used to automatically checkout the current branch):

```
git clone
git@git.dave.eu:dave/bora/bora.git -b bora
```

- Enter the git directory
- Switch to **bora** branch (not required if this is already the current branch): `git checkout bora`
- Set project directory variable:

  ```
  export PROJ_DIR=$
  (pwd)/../bora-build-YYYYMMDD-nobk
  ```

- Configure Vivado settings:

  ```
  . /opt/Xilinx/Vivado/2013.3/settings64.sh
  ```
  [1]

- Launch Vivado with *build_project* script:

  ```
  vivado -mode tcl -source build_project.tcl
  -notrace -tclargs "-bitstream"
  ```
  [2]

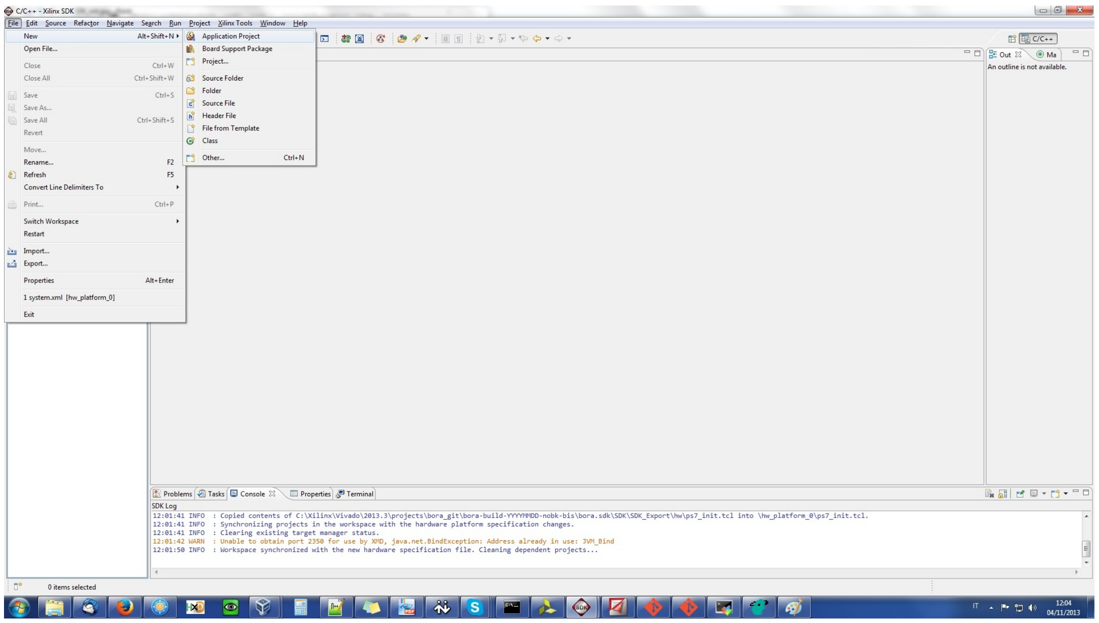
1 In a 32 bit system, Vivado settings are configured with the following command `/opt/Xilinx/Vivado/2013.3/settings32.sh`

2 Passing the -tclargs "-bitstream" parameters allows for automatic building of the FPGA bitstream.

3.2.2 FSBL

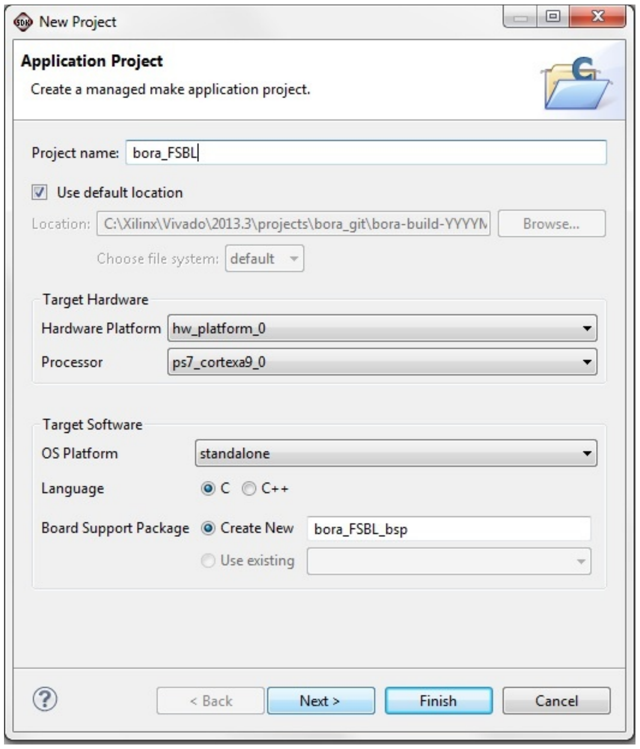
Once the Vivado project build is completed, the hardware configuration can be exported starting the SDK to build the FSBL. From the SDK GUI:

- Create a new application project, as shown in the picture below:



```
12:01:41 INFO  : Copied contents of C:\Xilinx\Vivado\2013.3\projects\bora_git\bora-build-YYYYMMDD-nobk-bis\bora.sdk\SDK\SDK_Export\hw\ps7_init.tcl into \hw_platform_0\ps7_init.tcl.
12:01:41 INFO  : Synchronizing projects in the workspace with the hardware platform specification changes.
12:01:41 INFO  : Clearing existing target manager status.
12:01:42 WARN  : Unable to obtain port 2350 for use by XMD, java.net.BindException: Address already in use: JVM_Bind
12:01:50 INFO  : Workspace synchronized with the new hardware specification file. Cleaning dependent projects...
```

- Configure the application settings as shown in the pictures below:



Project name: bora_FSBL

[✓] Use default location

Location: C:\Xilinx\Vivado\2013.3\projects\bora_git\bora-build-YYYYN

Choose file system: default

Target Hardware

Hardware Platform: hw_platform_0

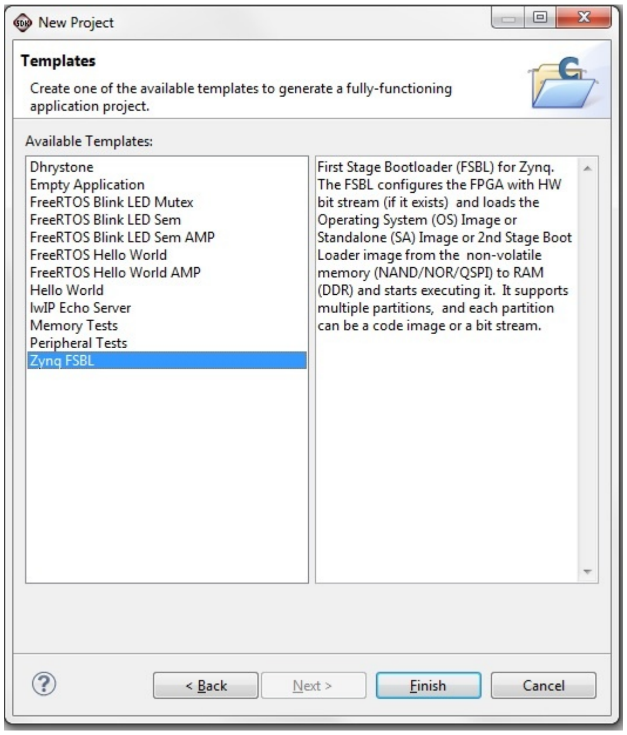
Processor: ps7_cortexa9_0

Target Software

OS Platform: standalone

Language: [✓ C] C++

Board Support Package: [✓ Create New] bora_FSBL_bsp; Use existing



First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream.

- Click finish to launch FSBL build process
- Create the binary from the FSBL ELF chosing one of the following options:
  - manually launch the command:
    ```
    arm-xilinx-eabi-objcopy -v -O binary
    $PROJ_DIR/bora.sdk/SDK/SDK_Export/bora_FSBL/D
    ebug/bora_FSBL.elf
    $PROJ_DIR/bora.sdk/SDK/SDK_Export/bora_FSBL/D
    ebug/bora_FSBL.bin
    ```
  - configure the automatic binary generation on project build. In Project Explorer, right-click on “bora_FSBL” project and select **C/C++ Build Settings** and add the command `arm-xilinx-eabi-objcopy -v -O binary ${ProjName}.elf ${ProjName}.bin` on **Post-build steps**

N.B. When the Vivado project is modified, the binary must be re-generated with the following command:

```
python fpga-bit-to-bin.py --flip
$PROJ_DIR/bora.runs/bora_run_impl/bora_design_wrapper.
bit
$PROJ_DIR/bora.runs/bora_run_impl/bora_design_wrapper.
bin
```

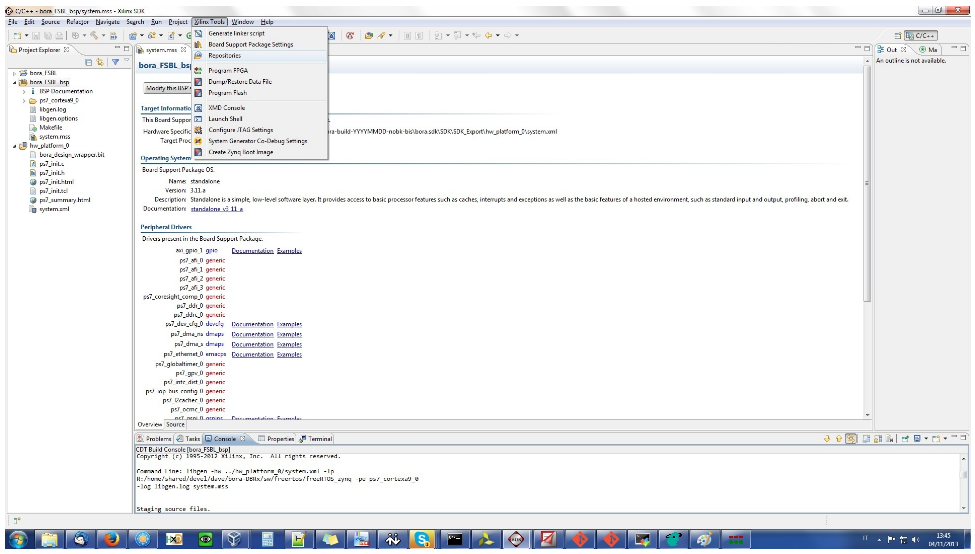
3.2.3 FreeRTOS applications

The following sections describe the steps required to configure and build both the Helloworld and the RPMsg-based examples.

3.2.3.1 Importing the FreeRTOS repository into the SDK

- Assuming that a local repository has not been created, clone the remote FreeRTOS git repository:
  ```
  git clone
  git@git.dave.eu:dave/bora/freertos.git
  ```
- Enter the git directory
- Switch to **freertos-AMP** branch: `git checkout freertos-AMP`

- In SDK gui import new repository: `Xilinx Tools -> Repositories`



bora_FSBL_bsp

Target Informatio

This Board Suppor

Hardware Specific ra-build-YYYYMMDD-nobk-bis\bora.sdk\SDK\SDK_Export\hw_platform_0\system.xml

Target Proc

Operating System

Board Support Package OS.

Name:	standalone
Version:	3.11.a
Description:	Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.
Documentation:	standalone_v3_11_a

Peripheral Drivers

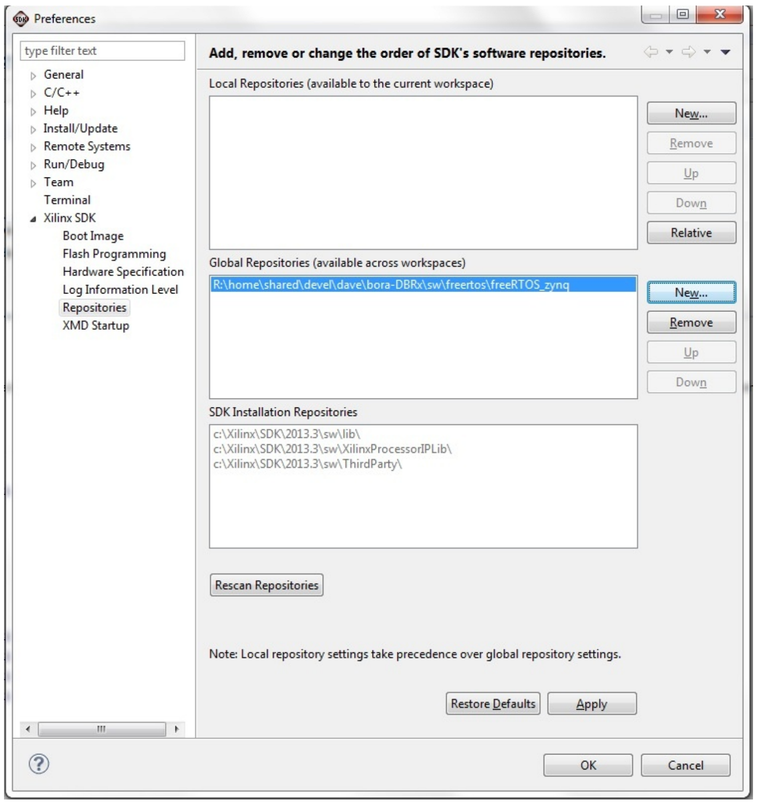
Drivers present in the Board Support Package.

axi_gpio_1	gpio	Documentation	Examples
ps7_afi_0	generic		
ps7_afi_1	generic		
ps7_afi_2	generic		
ps7_afi_3	generic		
ps7_coresight_comp_0	generic		
ps7_ddr_0	generic		
ps7_ddrc_0	generic		
ps7_dev_cfg_0	devcfg	Documentation	Examples
ps7_dma_ns	dmaps	Documentation	Examples
ps7_dma_s	dmaps	Documentation	Examples
ps7_ethernet_0	emacps	Documentation	Examples
ps7_globaltimer_0	generic		
ps7_gpv_0	generic		
ps7_intc_dist_0	generic		
ps7_iop_bus_config_0	generic		
ps7_l2cachec_0	generic		
ps7_ocmc_0	generic		

CDT Build Console [bora_FSBL_bsp]

```
Copyright (c) 1995-2012 Xilinx, Inc.  All rights reserved.

Command Line: libgen -hw ../hw_platform_0/system.xml -lp
R:/home/shared/devel/dave/bora-DBRx/sw/freertos/freeRTOS_zynq -pe ps7_cortexa9_0
-log libgen.log system.mss


Staging source files.
```

- Click New... to add a new repository under Local or Global Repositories, and select the freeRTOS repository directory:



Add, remove or change the order of SDK's software repositories.

Local Repositories (available to the current workspace)

Global Repositories (available across workspaces)

```
R:\home\shared\devel\dave\bora-DBRx\sw\freertos\freeRTOS_zynq
```

SDK Installation Repositories

```
c:\Xilinx\SDK\2013.3\sw\lib\
c:\Xilinx\SDK\2013.3\sw\XilinxProcessorIPLib\
c:\Xilinx\SDK\2013.3\sw\ThirdParty\
```

Note: Local repository settings take precedence over global repository settings.

- Click Rescan Repositories , Apply and OK
- At the end of the procedure, applications based on freeRTOS operating system can be built

3.2.3.2 Building Example #1: HelloWorld application

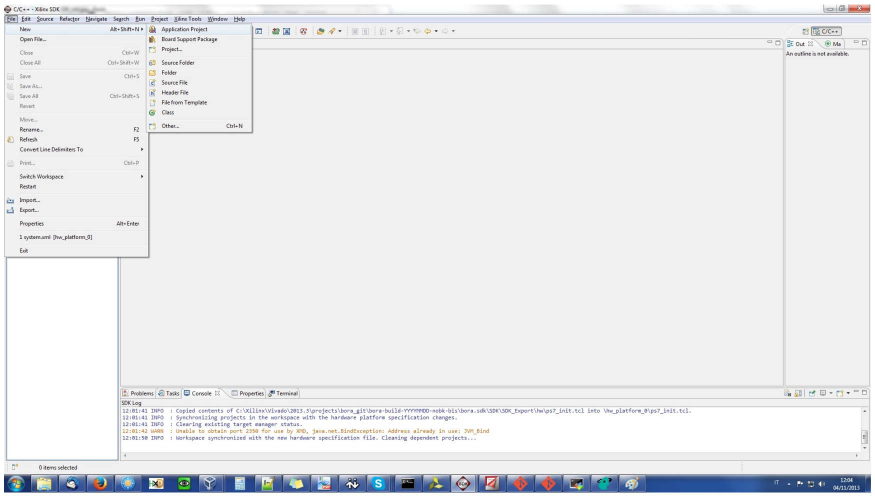
The first example shows basic AMP functionalities. On FreeRTOS side, UART0 is used to implement a simple console. This port is routed via EMIO signals to pin-strip connector of BoraEVB. Since these signals are driven by FPGA Bank #34, these pins are 3.3V. Thus a RS232 transceiver or an USB/UART bridge should be used in order to connect the console on a PC. The signals are routed to the JP17 connector of the BoraEVB as reported below:

- **JP17.4** – UART0_TX
- **JP17.6** – UART0_RX

Please follow the steps listed below to build a HelloWorld application that prints a message on UART0 (via EMIO) on

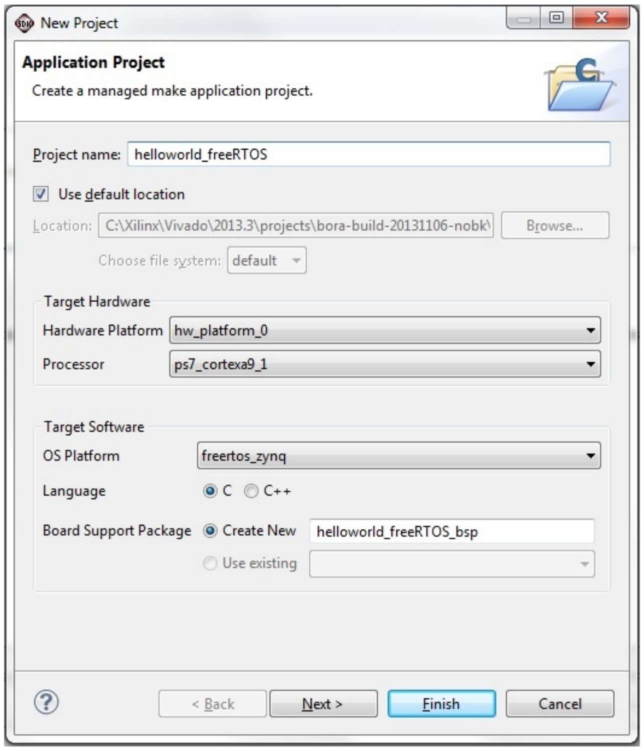
FreeRTOS running on Bora core #2.

- From the SDK GUI, create e new application project:



```
12:01:41 INFO  : Copied contents of C:\Xilinx\Vivado\2013.3\projects\bora_git\bora-build-YYYYMMDD-nobk-bis\bora.sdk\SDK\SDK_Export\hw\ps7_init.tcl into \hw_platform_0\ps7_init.tcl.
12:01:41 INFO  : Synchronizing projects in the workspace with the hardware platform specification changes.
12:01:41 INFO  : Clearing existing target manager status.
12:01:42 WARN  : Unable to obtain port 2350 for use by XMD, java.net.BindException: Address already in use: JVM_Bind
12:01:50 INFO  : Workspace synchronized with the new hardware specification file. Cleaning dependent projects...
```

- Configure the application settings as shown in the pictures and table below:



Project name: helloworld_freeRTOS

[✓] Use default location

Location: C:\Xilinx\Vivado\2013.3\projects\bora-build-20131106-nobk\

Choose file system: default

Target Hardware

Hardware Platform: hw_platform_0

Processor: ps7_cortexa9_1

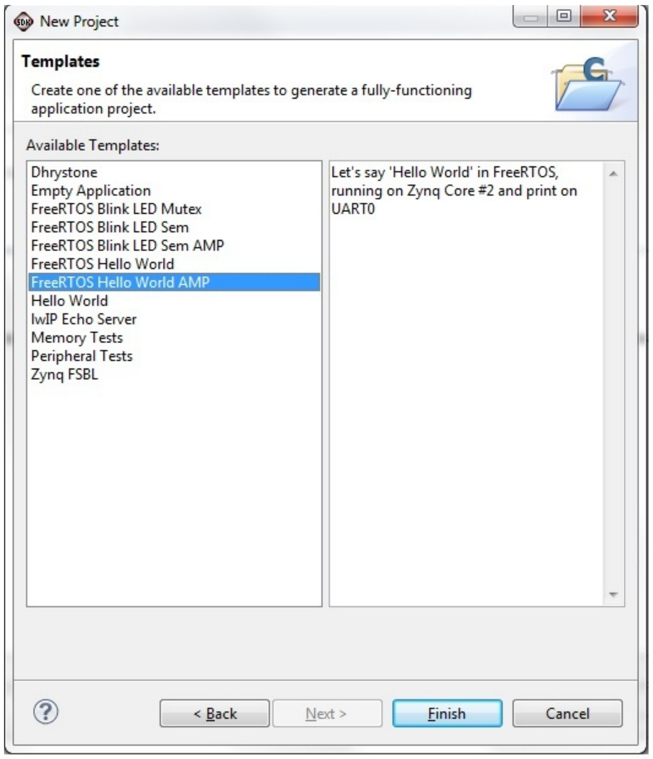
Target Software

OS Platform: freertos_zynq

Language: [✓ C] C++

Board Support Package: [✓ Create New] helloworld_freeRTOS_bsp

Use existing


New Project

Templates

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

Available Templates:

- Dhrystone
- Empty Application
- FreeRTOS Blink LED Mutex
- FreeRTOS Blink LED Sem
- FreeRTOS Blink LED Sem AMP
- FreeRTOS Hello World
- FreeRTOS Hello World AMP
- Hello World
- lwIP Echo Server
- Memory Tests
- Peripheral Tests
- Zynq FSBL

Let's say 'Hello World' in FreeRTOS, running on Zynq Core #2 and print on UART0

Project name	`helloworld_freeRTOS`
Hardware Platform	`hw_platform_0`
Processor	`ps7_cortexa9_1`
OS Plaftorm	`freertos_zynq`
Language	`C`
Board Support Package	`Create New`
Type	`FreeRTOS Hello World AMP template`

- Click finish to launch the application build process
- Create the binary from the application ELF chosing one of the following options:
  - manually launch the command:
    ```
    arm-xilinx-eabi-objcopy -v -O binary
    $PROJ_DIR/bora.sdk/SDK/SDK_Export/hellowordl_
    freeRTOS/Debug/hellowordl_freeRTOS.elf
    $PROJ_DIR/bora.sdk/SDK/SDK_Export/hellowordl_
    ```

`freeRTOS/Debug/hellowordl_freeRTOS.bin`

- configure the automatic binary generation on project build. In Project Explorer, right-click on "helloworld_freeRTOS" project and select **C/C++ Build Settings** and add the command `arm-xilinx-eabi-objcopy -v -O binary ${ProjName}.elf ${ProjName}.bin` on **Post-build steps**

3.2.3.3 Building Example #2: RPMsg-based application

The procedure needed to build this application is similar to the one used to build HelloWorld application. The only difference is that the `FreeRTOS Latency AMP` template must be selected.

In this case please note that:

- the standard Linux infrastructure will be used to load the firmware for the second core
- Linux will start in SMP mode, running on both cores; then CPU1 will be shutdown and freeRTOS firmware will be loaded and run

This application exploits TTC1 timer to measure IRQ latencies as described in Xilinx UG978. In addition to that, GPIO0 (pin JP21.16 on BoraEVB) will be toggled every time ISR is invoked.

Once the build process is completed, the executable file in .elf format will be generated (we suggest to name it `freertos`). Creating the .bin file is not required.

Project name	`RPMsg_freeRTOS`
Hardware Platform	`hw_platform_0`
Processor	`ps7_cortexa9_1`
OS Plaftorm	`freertos_zynq`
Language	`C`
Board Support Package	`Create New`
Type	`FreeRTOS Latency AMP`

To run this example, Linux kernel[3] must be rebuilt too[4]. First of all copy the freertos executable file in .elf format (`freertos`) into the directory `firmware` of Linux kernel tree[5]. Then configure the kernel using `bora_amp_defconfig` as configuration file and enter the following command line, that changes the default load address of kernel and launches the building of both the kernel image and the modules:

```
bash# make UIMAGE_LOADADDR=0x10008000 uImage modules
[...]
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-3.9.0-bora-1.1.0-xilinx-00
Created:      Thu Nov 21 15:55:07 2013
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    3217192 Bytes = 3141.79 kB = 3.07 MB
Load Address: 10008000
Entry Point:  10008000
  Image arch/arm/boot/uImage is ready
```

The file `arch/arm/boot/uImage` is the binary image of the kernel that must be used to boot the system.

The following kernel modules, resulting from the kernel build procedure, must be copied from the building directory to the root file system (usually into `/lib/modules/<kernel version>/kernel`, but any other directory can be used):

```
  LD [M]  drivers/remoteproc/remoteproc.ko
  LD [M]  drivers/remoteproc/zynq_remoteproc.ko
  LD [M]  drivers/rpmsg/rpmsg_freertos_statistic.ko
  LD [M]  drivers/rpmsg/virtio_rpmsg_bus.ko
  LD [M]  drivers/virtio/virtio.ko
  LD [M]  drivers/virtio/virtio_ring.ko
  LD [M]  net/rpmsg/rpmsg_proto.ko
```

For further details on kernel modules, please refer to http://tldp.org/HOWTO/Module-HOWTO/

3.2.4 Linux Device Tree

The Flattened Device Tree (FDT) is a data structure for describing the hardware in a system (for further information, please refer to http://elinux.org/Device_Tree). Both Example #1

3 The kernel branch must be bora.
4 It is assumed that the development environment is already set up as described in BELK Quick Start Guide.
5 The name of the binary file copied into the `firmware` directory must be `freertos`

and Example #2 requires some modifications to the standard Bora device tree (to initialiaze UART0 port and to properly initialize the RPMsg infrastructure, respectively). Please use the kernel branch **bora**, that already includes the aforementioned patches (for further details, please refer to the `arch/arm/boot/dts/bora.dts` file and commit descriptions on the Linux git repository). For detailed instructions on how to build the Linux kernel and the Device Tree, please refer to the BELK Quick Start Guide (Section 3.4.3).

3.3 Running the demo applications

3.3.1 Example #1: HelloWorld application

This section describes how to run freeRTOS HelloWorld example application on BORA using AMP (Linux + FreeRTOS). Plese follow the steps listed below:

- Place all the binary files into the host tftp directory:

Kernel[6]	`uImage`
Device Tree	`bora.dtb`
First stage bootloader	`bora_FSBL.bin`
FPGA bitstream	`bora_design_wrapper.bin`
FreeRTOS application	`helloworld_freeRTOS.bin`

- Start the Bora system
- From the U-Boot shell, update the FSBL with the following commands:

```
run load_fsbl
run update_fsbl
```

- Reset the board to reboot with the new FSBL
- Add the following U-Boot environment variables[7]:

```
setenv addcons 'setenv bootargs ${bootargs} console=$
{console},115200n8 cma=16M debug maxcpus=${nr_cpus}'
setenv addmem 'setenv bootargs ${bootargs} mem=$
(kernel_mem)'
setenv kernel_mem 1008M
setenv nr_cpus 1
setenv net_nfs 'run program_fpga; run load_freertos;
run loadk nfsargs addip addcons addmem; bootm $
{loadaddr_kern} - ${loadaddr_ftd}'
setenv load_freertos 'tftp ${freertos_addr} $
{freertos_file};mw.l 0xFFFFFFF0 ${freertos_addr}'
```

6 The kernel must be built with the UIMAGE_LOADADDR 0x8000 option. Please refer to section 3.4.3 of the Belk Quick Start Guide

7 program_fpga: Loads FPGA binary from TFTP and programs the bitstream
load_freertos: Loads freertos application binary from TFTP and writes application start address for core#2
mem=${kernel_memory}: sets maximum kernel memory (1008M = 1024M - 16M)
maxcpus=${nr_cpus}: sets maximum Linux cpus to 1

```
setenv freertos_addr 0x3F000000
setenv freertos_file bora/BELK/helloworld_freeRTOS.bin
setenv fpga_file BELK/bora_design_wrapper.bin
```

- Boot the system running the following command:

  `run net_nfs`

3.3.2 Example #2: RPMsg-based application

As stated before, this example shows a more sophisticated approach that allows for:

- using a standardized communication channel between the two cores
- exploiting a standardized mechanism to load the firmware of second core

The example performs IRQ latency measurements on FreeRTOS side by using a hardware timer. These measures are collected by the counterpart application running on Linux side and shown on console. Plese follow the steps listed below:

- Place all the binary files into the host tftp directory:

Kernel	uImage
Device Tree	bora.dtb
First stage bootloader	bora_FSBL.bin
FPGA bitstream	bora_design_wrapper.bin
FreeRTOS application	freertos

- Start the Bora system
- From the U-Boot shell, update the FSBL with the following commands:

  `run load_fsbl`

  `run update_fsbl`

- Reset the board to reboot with the new FSBL
- Add the following U-Boot environment variables[8][9]:

8 program_fpga: Loads FPGA binary from TFTP and programs the bitstream
load_freertos: Loads freertos application binary from TFTP and writes application start address for core#2
9 Please note that, using the RPMsg mechanism, it's not required to set the maxcpus=${nr_cpus} variable

```
setenv addcons 'setenv bootargs ${bootargs} console=$
{console},115200n8 cma=16M debug'
setenv addmem 'setenv bootargs ${bootargs} mem=$
(kernel_mem)'
setenv kernel_mem 496M
setenv net_nfs 'run program_fpga; run loadk nfsargs
addip addcons addmem; bootm ${loadaddr_kern} - $
{loadaddr_ftd}'
setenv freertos_addr 0x3F000000
setenv fpga_file BELK/bora_design_wrapper.bin
```

- Boot the system running the following command:

  ```
  run net_nfs
  ```

When booting, the Linux kernel will print out the following message to indicate it has been relocated to address 0x10000000:

```
[    0.000000] Machine: Xilinx Zynq Platform, model:
Bora
[    0.000000] Change memory bank to 10000000-2fffffff
[    0.000000] cma: CMA: reserved 16 MiB at 2f000000
```

To start the example, please enter the following commands on Linux side to load the required modules:

```
insmod  drivers/virtio/virtio.ko
insmod  drivers/virtio/virtio_ring.ko
insmod  drivers/rpmsg/virtio_rpmsg_bus.ko
insmod  net/rpmsg/rpmsg_proto.ko
insmod  drivers/remoteproc/remoteproc.ko
insmod  drivers/remoteproc/zynq_remoteproc.ko
insmod  drivers/rpmsg/rpmsg_freertos_statistic.ko
```

Linux kernel will print these messages, informing that the communication between the two cores has been established:

```
[   17.966158] NET: Registered protocol family 41
[   18.036698] CPU1: shutdown
[   18.045287]  remoteproc0: 0.remoteproc-test is
available
[   18.050522]  remoteproc0: Note: remoteproc is still
under development and considered experimental.
[   18.059554]  remoteproc0: THE BINARY FORMAT IS NOT
YET FINALIZED, and backward compatibility isn't yet
guaranteed.
[   18.077341]  remoteproc0: powering up
0.remoteproc-test
[   18.082668]  remoteproc0: Booting fw image freertos,
size 2357682
[   18.103607]  remoteproc0: remote processor
0.remoteproc-test is now up
[   18.113339] virtio_rpmsg_bus virtio0: rpmsg host is
```

```
online
[   18.118795]  remoteproc0: registered virtio0 (type
7)
[   18.124417] virtio_rpmsg_bus virtio0: creating
channel rpmsg-timer-statistic addr 0x50
[   18.151586] rpmsg_freertos_statistic rpmsg0: new
channel: 0x400 -> 0x50!
```

Then run the `latencystat` application as shown below. The typical output will look like this:

```
root@bora:~# ./latencystat -b
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 323 ns (36 ticks) had 38 frequency
        Bucket 341 ns (38 ticks) had 299 frequency
        Bucket 512 ns (57 ticks) had 1 frequency
        Bucket 746 ns (83 ticks) had 1 frequency
-------------------------------------------------------
----
Histogram Data:
        min: 323 ns (36 ticks)
        avg: 332 ns (37 ticks)
        max: 746 ns (83 ticks)
        out of range: 0
        total samples: 339
-------------------------------------------------------
----
```

This application is extremely useful for evaluating how CPU load on first core affects IRQ latency. In case latency does not satisfy real-time requirements, it may be necessary to adjust arbitration priorities of processor's interconnect subsystem. For further details, please refer to chapter “Interconnect” of Zynq Reference Manual.

N.B. prior to launching the `latencystat` application, make sure that the governor is set to "performance" with the following command:

```
echo performance >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

4 Advanced debugging techniques for AMP Linux+FreeRTOS configuration

4.1 Introduction

When working with complex real-time configurations such as AMP Linux+FreeRTOS, debugging requirements increase dramatically. This chapter – written in collaboration with Lauterbach SRL (www.lauterbach.it) – shows how these issues can be tackled with Lauterbach TRACE32 ® debugger[10].

The following picture shows the BoraEVB connected to Lauterbach PowerDebug Interface/USB3 via J18 connector. By default, the board is configured to chain Xilinx PL TAP and ARM DAP (please refer to chapter "JTAG and DAP Subsystem"

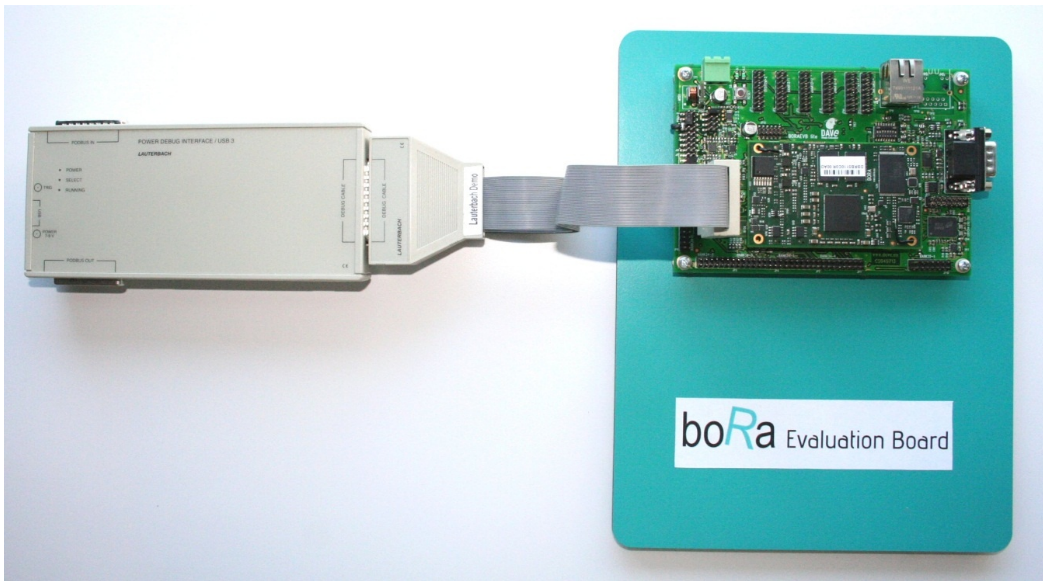


Fig. 3: BoraEVB connected to Lauterbach PowerDebug Interface/USB3

10 The techniques described in this chapter apply to the "Example #1: HelloWorld" FreeRTOS application (please refer to section 3.2.3.2).

of Zynq Reference Manual for more details).

The following sections describe in detail how to configure TRACE32 ® debugger to support debug of Linux running on the first Zynq core, and FreeRTOS, running on the second Zynq core.

4.2 Prerequisites

- LA-3500 Power Debug USB3 or LA-7705 Power Debug Ethernet or LA-7699 PowerDebug II
- LA-7843 JTAG Debugger for Cortex-A/-R
- LA-7960X License for Multicore Debugging
- TRACE32 PowerView for ARM (Release: Feb 2013, Software Version: R.2013.02.000045901)
- Optional: LA-7970X Trace License for ARM (Debug Cable)

For a general introduction to debug features provided by TRACE32 tools, please refer to:

- “Debugger Basics – Training” manual (training_debugger.pdf)
- “Training HLL Debugging” manual (training_hll.pdf)

4.3 TRACE32 configuration

In AMP configuration, each core runs a unique code, already fixed at compile time. The CPU interoperates with other processing units, exchanging data through dedicated channels (for example, shared memory buffers or peripheral units). Lauterbach supports these architectures with different TRACE32 instances, each one connected to a single core, in “core view” configuration where debug focus is on single processor.

However, as the cores do not work independently but perform the application task together and in parallel, it is possible to start and stop all the cores simultaneously. This is the only way to test the interaction between the cores and to monitor and control the entire application. Moreover, as each core run a separate part of the application, the majority of the symbol and debug information is assigned exclusively to the corresponding core.

In the following paragraphs, the basic TRACE32 multicore configuration for a single device will be introduced. For more details, please refer to “ICD Debugger User's Guide” (debugger_user.pdf).

4.3.1 Multicore configuration

For the configuration of TRACE32 application, reference scripts are provided from Lauterbach. The first GUI must be started manually and must register itself to share a common JTAG handler with other TRACE32 applications. This is done setting the option `CORE=` in the configuration file (default file name: `config.t32`).

```
PBI=       ; within config file of first core
USB
CORE=1

or:

PBI=       ; within config file of first core
NET
NODE=<IP_address>
PACKLEN=1024
CORE=1
```

Nevertheless, the setting to define which core is addressed, actually is done later on.

4.3.2 Multicore synchronization

To use the start/stop synchronization between different core debuggers, the INTERCOM port settings are necessary. This is done assigning predefined port numbers in the configuration file to each TRACE32 application (option PORT=).

```
IC=NETASSIST          ; within config file of
first core
PORT=20001
```

```
IC=NETASSIST        ; within config file of
second core
PORT=20002
```

4.4 Startup scripts

In order to use a generic configuration file for each TRACE32 instance, there is the possibility to use just one generic template file for all cores. The particular settings are passed as parameter. This is shown in the reference script:

```
amp_start_core0.bat
```

which refers to the configuration file:

```
amp_config.t32
```

The batch file starts automatically this startup script:

```
amp_demo.cmm
```

After booting the first TRACE32 GUI, the second GUI will be started automatically by the startup script. See the reference script:

```
amp_demo_start_core1.cmm
```

For more details about PRACTICE batch language, please refer to:

- “Training PRACTICE” manual (training_practice.pdf)
- “PRACTICE Script Language User´s Guide” (practice_user.pdf)
- “PRACTICE Script Language Reference Guide” (practice_ref.pdf)

4.4.1 PRACTICE macros for multiple TRACE32

The startup script, started automatically at the first TRACE32 application, is fully able to configure the whole debug system, providing PRACTICE commands both to the current instance of TRACE32 application, and to the second instance. It’s also possible to deliver the same PRACTICE command to both instances with a single command line. The command redirection is possible using the INTERCOM feature. Typically some PRACTICE macros can be defined for this purpose.

```
&core0=""                                  ;only to
improve readability
&core1="intercom
localhost:&intercomport_core1"
&both="GOSUB intercom_both "
```

where:

```
intercom_both:
  LOCAL &param
  ENTRY %Line &param
  &core0 &param
  &core1 &param
RETURN
```

In this way, all CPU-specific configuration commands can be performed in the same way for each TRACE32 application, or distinguishing between different configurations. For example:

```
&both SYStem.RESet
```

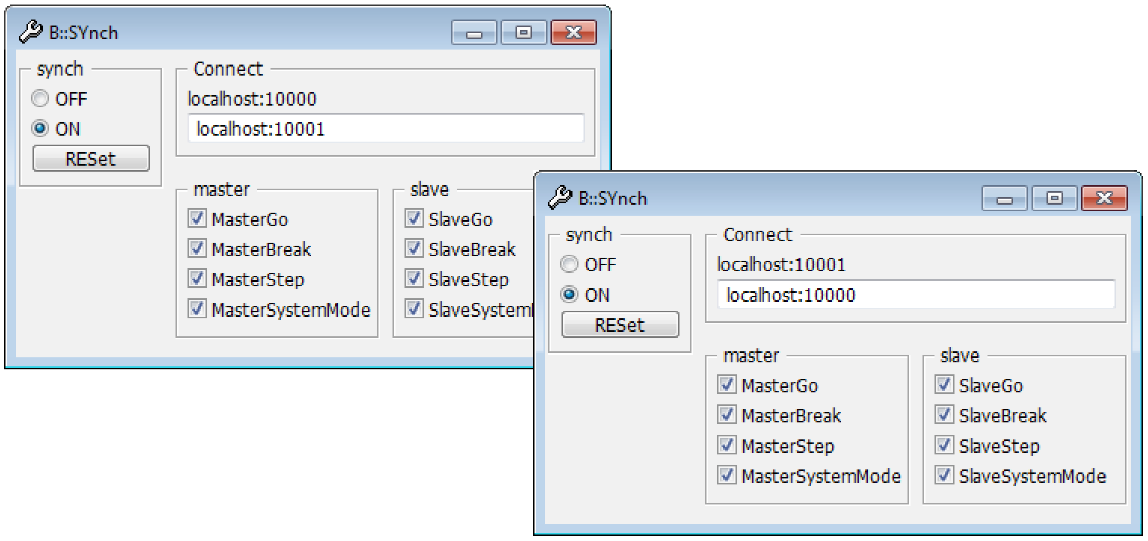
or:

```
&core0 SYStem.CPU ZYNQ-7000CORE0

&core1 SYStem.CPU ZYNQ-7000CORE1
```

4.4.2 The SYnch command

The synchronization between different TRACE32 applications is done by SYnch command group, which allows the following purposes:

- to establish a start/stop synchronization between the cores controlled by different TRACE32 instances;
- to allow concurrent assembler single steps between the cores controlled by different TRACE32 instances;
- to allow synchronous system mode changes between the cores controlled by different TRACE32 instances.



B::SYnch

synch: OFF [✓ ON] RESet

Connect: localhost:10000 localhost:10001

master: ✓ MasterGo ✓ MasterBreak ✓ MasterStep ✓ MasterSystemMode

slave: ✓ SlaveGo ✓ SlaveBreak ✓ SlaveStep ✓ SlaveSystem

B::SYnch

synch: OFF [✓ ON] RESet

Connect: localhost:10001 localhost:10000

master: ✓ MasterGo ✓ MasterBreak ✓ MasterStep ✓ MasterSystemMode

slave: ✓ SlaveGo ✓ SlaveBreak ✓ SlaveStep ✓ SlaveSystemMode

The SYnch settings are reported below:

Connect

Establish a connection to the debugger attached to the defined communication port(s). Several debuggers ports can be specified, separated by space.

MasterGo ON

If the program execution is started, the program execution for all other processors which have SlaveGo ON is also started.

MasterBrk ON

If the program execution is stopped, the program execution for all other debuggers which have SlaveBrk ON is also stopped.

MasterStep ON

If an asm single step is executed, all processors which have SlaveStep ON will also asm single step.

MasterSystemMode ON

Invite other TRACE32 instances to perform system mode changes synchronously. System mode changes are typically performed by the commands SYStem.Mode <mode>

SlaveGo ON

The program execution is started, if a processor with MasterGo ON starts its program execution.

SlaveBrk ON

The program execution is stopped, if a processor with MasterBrk ON stops its program execution.

SlaveStep ON

A asm single step is performed, if a processor with MasterStep On performs an asm single step.

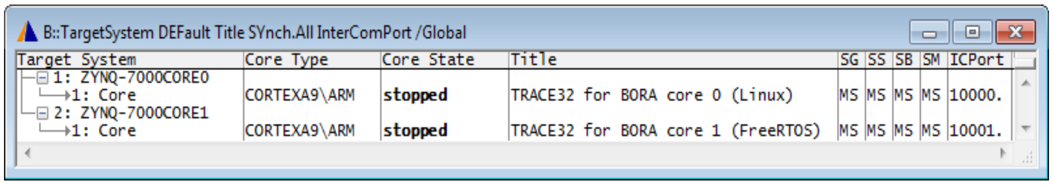
SlaveSystemMode ON

Synchronize with system mode changes in connected TRACE32 instances.

A summary of SYnch configuration is provided with command `TargetSystem`, which also allows to easily and rapidly modify the SYnch mode options.

Moreover the `TargetSystem` command provides a general overview of the whole multicore configuration and of the current state.

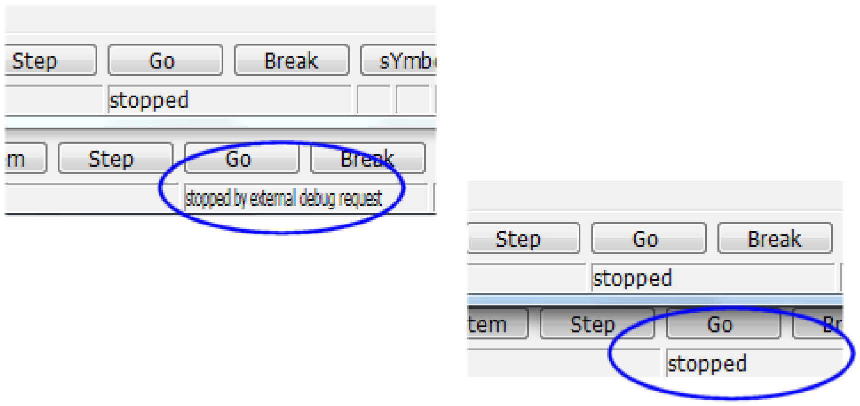
```
TargetSystem DEFault Title SYnch.All
InterComPort /Global
```



B::TargetSystem DEFault Title SYnch.All InterComPort /Global

Target System	Core Type	Core State	Title	SG	SS	SB	SM	ICPort
1: ZYNQ-7000CORE0								
1: Core	CORTEXA9\ARM	**stopped**	TRACE32 for BORA core 0 (Linux)	MS	MS	MS	MS	10000.
2: ZYNQ-7000CORE1								
1: Core	CORTEXA9\ARM	**stopped**	TRACE32 for BORA core 1 (FreeRTOS)	MS	MS	MS	MS	10001.

There is a time delay between reaction of different cores. The



reaction time of the slave core depends on the technical realisation of the synchronization. If no specific configuration is performed, the synchronization is done by software (eg: the master debugger informs the slave debugger via socket communication on the host about specified events). On the other side, if the on-chip Cross Trigger Interface is configured, the synchronization takes place directly on the processor. This is a faster solution and the time delay between reaction of different cores becomes around 0-10ns.

The CTI interface for chip Zynq is configured with the following commands. If CTI is configured, TRACE32 will use the faster solution for the synchronization.

```
&core0 SYStem.CONFIG.CTI.Base 0x80098000

&core0 SYStem.CONFIG.CTI.Config CORTEXV1
```

```
&core1 SYStem.CONFIG.CTI.Base 0x80099000

&core1 SYStem.CONFIG.CTI.Config CORTEXV1
```

4.5 Setting up the Linux debug configuration

The symbolic information is useful for HLL debugging, or setting breakpoints, stepping through the code, viewing variables, and many other aspects of debugging. The compiler must be configured in order to generate debug symbols. The `vmlinux` file for the running kernel must be available, in order to load the kernel debug symbols.

No instrumentation is needed in the kernel source code for debugging with Lauterbach, but it’s important that the `vmlinux` file is generated from the same kernel build as the `zImage` or `uImage` running on the system.

The `Data.LOAD` command is used to load the kernel symbols, and the `sYmbol.SourcePATH` command can be used, if necessary, to define additional search directories for the source files. The `Data.LOAD` command for `vmlinux` is applied to `&core0`, so that symbol information is not shared with the other TRACE32 application.

Specific options must be configured to avoid automatic Break of TRACE32 debugger, in case any of the following events happens due to normal Linux operations.

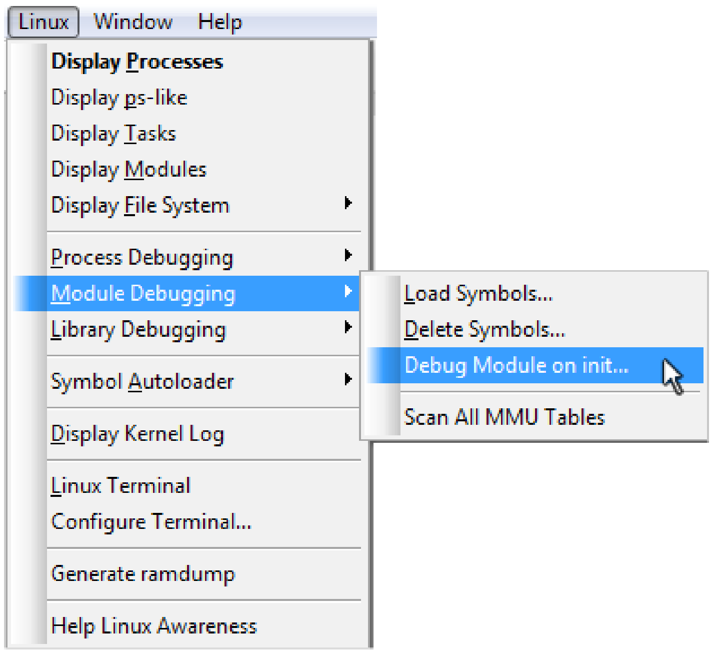
```
TrOnchip.Set UNDEF OFF    ; may be used by
Linux for FPU detection

TrOnchip.Set DABORT OFF   ; used by Linux for
page miss!

TrOnchip.Set PABORT OFF   ; used by Linux for
page miss!
```

In the following paragraphs, the basic TRACE32 Linux configuration will be introduced. For more details, please refer to:

- “Training Linux Debugging” manual (training_rtos_linux.pdf)
- “RTOS Debugger for Linux - Stop Mode” manual (rtos_linux_stop.pdf).

4.5.1 Kernel awareness

TRACE32 kernel awareness technology makes debugger aware



of the OS running in the target system. Debug is significantly simplified, as the user can immediately access all the components of the OS and the application. The Executable and Linkable Format (ELF) binaries, created at kernel build time, are used also by Linux awareness.

The Linux kernel awareness is configured with the commands:

```
; loads Linux awareness

&core0 TASK.CONFIG
~~/demo/arm/kernel/linux/linux-3.x/linux3.t32

; loads Linux menu

&core0 MENU.ReProgram
~~/demo/arm/kernel/linux/linux-3.x/linux.men
```

The Linux kernel awareness is loaded into the first TRACE32 application, and will not affect the second one.

The Linux menu file (`linux.men`) includes many useful menu items developed for the TRACE32 GUI to ease Linux debugging.

4.5.2 MMU support

In Linux embedded, the Lauterbach debuggers provide a very tight integration with the RTOS. The kernel awareness supports Linux MMU format and is able to handle virtual memory addressing.

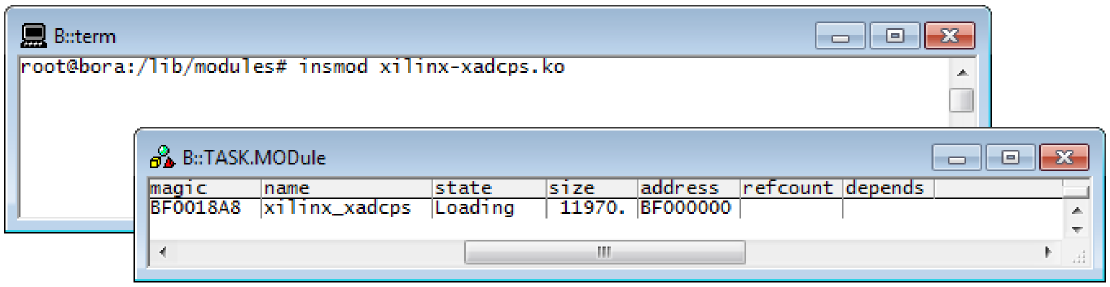
```
MMU.FORMAT LINUX swapper_pg_dir
0xc0000000--0xc1ffffff 0x00000000

TRANSlation.Create 0xc0000000--0xc1ffffff
0x00000000

TRANSlation.COMMON 0xbf000000--0xffffffff

TRANSlation.TableWalk ON

TRANSlation.ON
```

where the virtual address range is the virtual-to-physical kernel address mapping, according with results of command `MMU.List KernelPageTable`, executed when the kernel is up and running.

In the command TRANSlation.COMMON, the virtual address range has been extended below the kernel start address, because kernel objects are loaded in this memory range.

4.5.3 Debugging of kernel modules

The Linux kernel can be compiled to allow linking of additional modules at runtime (kernel objects). The Lauterbach debuggers also support kernel modules debugging, starting from the initialization function.



```
root@bora:/lib/modules# insmod xilinx-xadcps.ko
```

```
magic     name          state    size    address  refcount depends
BF0018A8  xilinx_xadcps Loading   11970. BF000000
```

4.5.4 Debugging of user processes, threads, shared objects

User process debugging is also available, starting from the very beginning of the process. If the process loads shared objects, they are loaded in the process address space when the related instructions are executed for the first time (demand paging).

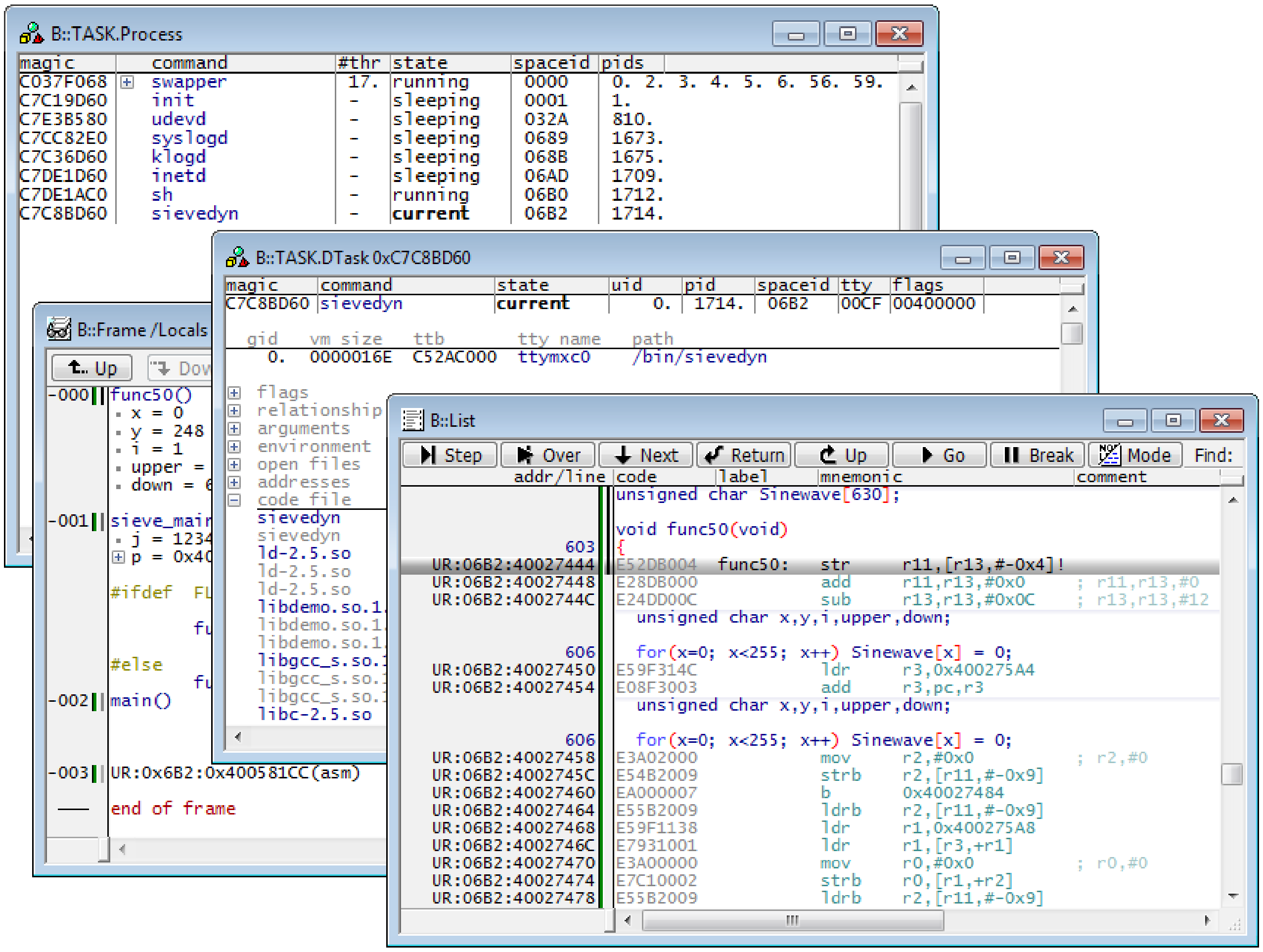
The Lauterbach debuggers also support debug of threads for multithreaded processes. In this case, the same address space is shared between different threads and the symbolic information can be loaded only once per process.

In general, the same techniques used for debugging kernel code, such as setting breakpoints, stepping through code, watching variables, and viewing memory contents, can be performed in the same way for processes and tasks.

Virtual address spaces are distinguished in TRACE32 using the concept of spaceID, which is enabled with the option:

```
SYStem.Option MMUspaces ON
```

The memory addressing is extended using the lower 16 bit of the process PID, allowing in this way to distinguish between equal virtual addresses for different processes.



```
magic     command     #thr  state     spaceid  pids
C037F068  swapper     17.   running   0000     0. 2. 3. 4. 5. 6. 56. 59.
C7C19D60  init        -     sleeping  0001     1.
C7E3B580  udevd       -     sleeping  032A     810.
C7CC82E0  syslogd     -     sleeping  0689     1673.
C7C36D60  klogd       -     sleeping  068B     1675.
C7DE1D60  inetd       -     sleeping  06AD     1709.
C7DE1AC0  sh          -     running   06B0     1712.
C7C8BD60  sievedyn    -     current   06B2     1714.
```

```
magic     command   state    uid  pid    spaceid  tty   flags
C7C8BD60  sievedyn  current   0.  1714.  06B2     00CF  00400000

  gid   vm size   ttb       tty name   path
   0.   0000016E  C52AC000  ttymxc0    /bin/sievedyn

⊞ flags
⊞ relationship
⊞ arguments
⊞ environment
⊞ open files
⊞ addresses
⊟ code file
   sievedyn
   sievedyn
   ld-2.5.so
   ld-2.5.so
   ld-2.5.so
   libdemo.so.1.
   libdemo.so.1.
   libdemo.so.1.
   libgcc_s.so.
   libgcc_s.so.
   libgcc_s.so.
   libc-2.5.so
```

```
-000  func50()
       x = 0
       y = 248
       i = 1
       upper =
       down = 6

-001  sieve_mair
       j = 1234
      ⊞ p = 0x40

      #ifdef  FL
              fu

      #else
              fu
-002  main()


-003  UR:0x6B2:0x400581CC(asm)

 ——   end of frame
```

```
          addr/line  code      label    mnemonic                       comment
                     unsigned char Sinewave[630];

                     void func50(void)
                603  {
UR:06B2:40027444     E52DB004  func50:  str      r11,[r13,#-0x4]!
UR:06B2:40027448     E28DB000           add      r11,r13,#0x0          ; r11,r13,#0
UR:06B2:4002744C     E24DD00C           sub      r13,r13,#0x0C         ; r13,r13,#12
                       unsigned char x,y,i,upper,down;

                606    for(x=0; x<255; x++) Sinewave[x] = 0;
UR:06B2:40027450     E59F314C           ldr      r3,0x400275A4
UR:06B2:40027454     E08F3003           add      r3,pc,r3
                       unsigned char x,y,i,upper,down;

                606    for(x=0; x<255; x++) Sinewave[x] = 0;
UR:06B2:40027458     E3A02000           mov      r2,#0x0               ; r2,#0
UR:06B2:4002745C     E54B2009           strb     r2,[r11,#-0x9]
UR:06B2:40027460     EA000007           b        0x40027484
UR:06B2:40027464     E55B2009           ldrb     r2,[r11,#-0x9]
UR:06B2:40027468     E59F1138           ldr      r1,0x400275A8
UR:06B2:4002746C     E7931001           ldr      r1,[r3,+r1]
UR:06B2:40027470     E3A00000           mov      r0,#0x0               ; r0,#0
UR:06B2:40027474     E7C10002           strb     r0,[r1,+r2]
UR:06B2:40027478     E55B2009           ldrb     r2,[r11,#-0x9]
```

4.6 Setting up the FreeRTOS debug configuration

A similar kernel awareness concept as Linux, is provided by Lauterbach for many other RTOS. Among the others, also FreeRTOS is supported with the following awareness configuration.

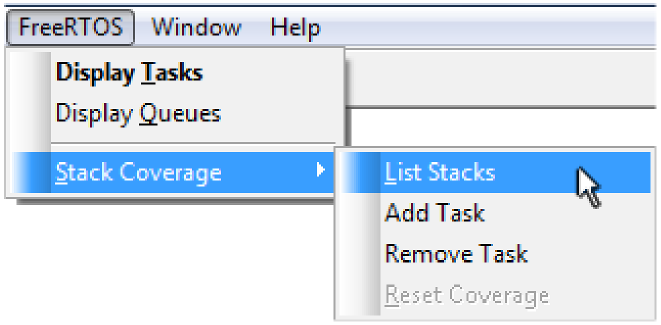
```
; load FreeRTOS awareness

&core1 TASK.CONFIG
~~/demo/arm/kernel/freertos/freertos.t32
```

```
; load FreeRTOS menu

&core1 MENU.ReProgram
~~/demo/arm/kernel/freertos/freertos.men
```

The FreeRTOS kernel awareness is loaded into the second TRACE32 application, and will not affect the first one.

As of symbol information it's enough to load the proper symbol file, redirecting the `Data.LOAD command` to `&core1`. In this way, the second TRACE32 application will have visibility of FreeRTOS HLL debug info.

The FreeRTOS menu file (`freertos.men`) includes useful menu



items developed for the TRACE32 GUI to ease FreeRTOS debugging.

For more details, please refer to "RTOS debugger for FreeRTOS" manual (rtos_freertos.pdf).

4.7 Onchip trace

The Zynq chip implements the ETB (Embedded Trace Buffer), a CoreSight hardware component providing on-chip trace functionality. The ETB stores program-flow trace information on-chip at high rates and at 32-bit data width. The data can be read out via JTAG, when the trace recording has ended.

This trace method is enabled in TRACE32 with the following commands, respectively performed on each GUIs of AMP configuration:

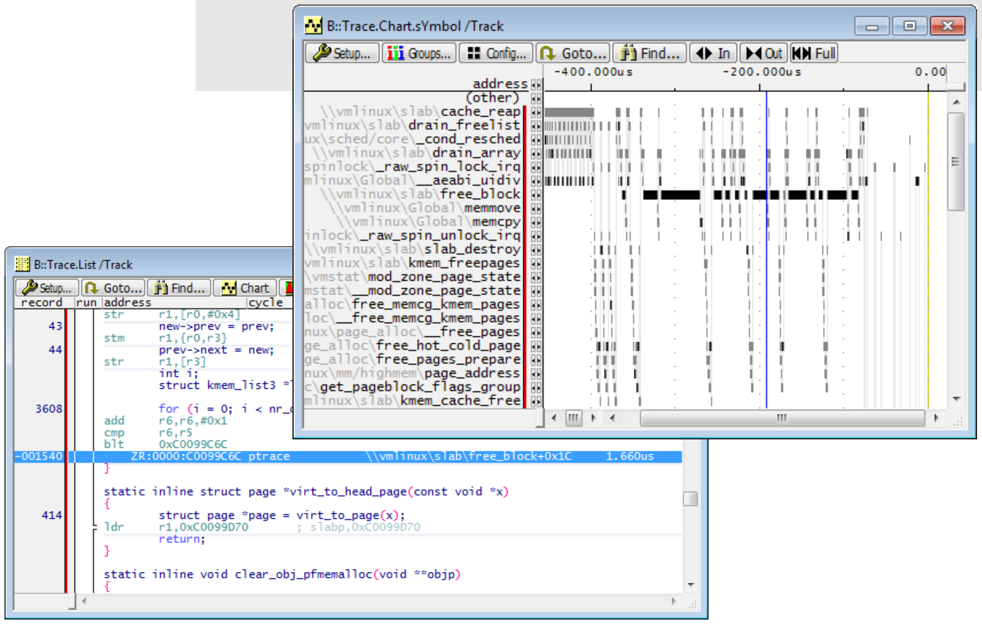
```
Trace.METHOD Onchip     ; select the ETB as
source                  ; for the trace
information
```

The TRACE32 menu item “Trace ? Configuration” allows a full control of configuration, initialization and listing of trace information.

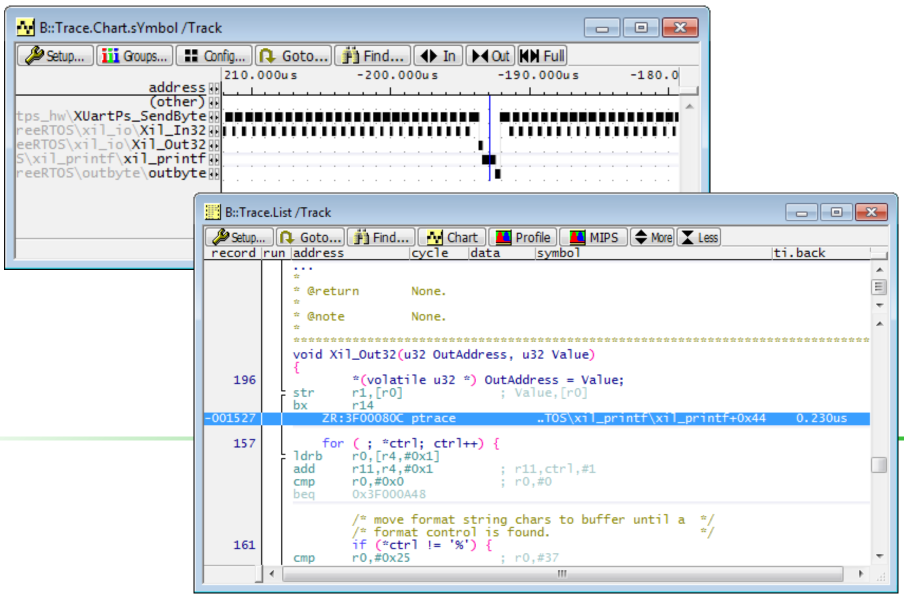
The trace, recorded for each core by each AMP TRACE32 GUI, can be synchronized using the XTRACK feature of TRACE32. In this way, an easy comparison is possible of program flow of each core at the same time.

The XTRACK feature is enabled with the following commands:

```
&core0 Synch.XTRACK
localhost:&intercomport_core1
```


B::Trace.Chart.sYmbol /Track
Setup... Groups... Config... Goto... Find... In Out Full
-400.000us -200.000us 0.00
address
(other)
\\vmlinux\slab\cache_reap
vmlinux\slab\drain_freelist
ux\sched/core_cond_resched
\\vmlinux\slab\drain_array
spinlock_raw_spin_lock_irq
mlinux\Global\__aeabi_uidiv
\\vmlinux\slab\free_block
\\vmlinux\Global\memmove
\\vmlinux\Global\memcpy
inlock_raw_spin_unlock_irq
\\vmlinux\slab\slab_destroy
vmlinux\slab\kmem_freepages
\vmstat\mod_zone_page_state
mstat__mod_zone_page_state
alloc\free_memcg_kmem_pages
loc__free_memcg_kmem_pages
nux\page_alloc\__free_pages
ge_alloc\free_hot_cold_page
ge_alloc\free_pages_prepare
nux\mm/highmem\page_address
c\get_pageblock_flags_group
mlinux\slab\kmem_cache_free
B::Trace.List /Track
Setup... Goto... Find... Chart

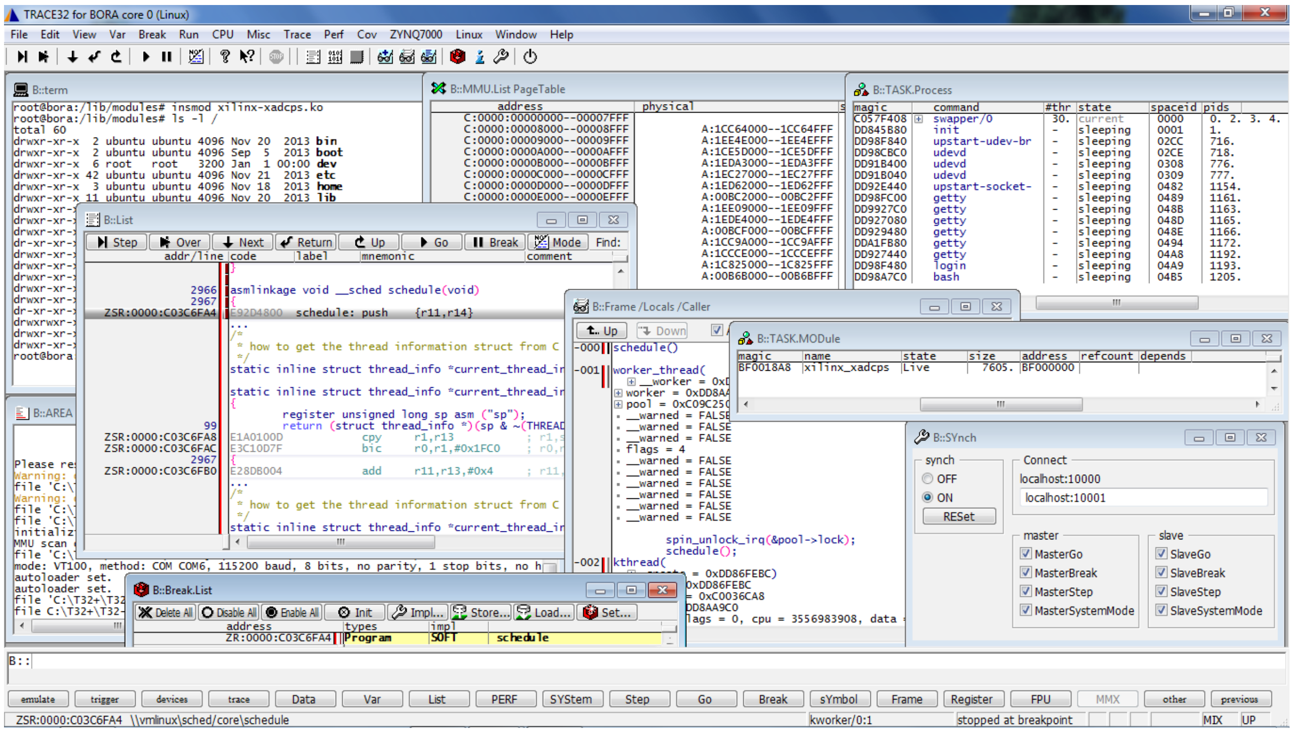
```
record  run address                                        cycle
            str       r1,[r0,#0x4]
    43                new->prev = prev;
            stm       r1,{r0,r3}
    44                prev->next = new;
            str       r1,[r3]
                      int i;
                      struct kmem_list3 *
  3608                for (i = 0; i < nr_
            add       r6,r6,#0x1
            cmp       r6,r5
            blt       0xC0099C6C
-001540         ZR:0000:C0099C6C ptrace               \\vmlinux\slab\free_block+0x1C     1.660us
            }

            static inline struct page *virt_to_head_page(const void *x)
            {
   414                struct page *page = virt_to_page(x);
            ldr       r1,0xC0099D70          ; slabp,0xC0099D70
                      return;
            }

            static inline void clear_obj_pfmemalloc(void **objp)
            {
```


B::Trace.Chart.sYmbol /Track
Setup... Groups... Config... Goto... Find... In Out Full
210.000us -200.000us -190.000us -180.0
address
(other)
tps_hw\XUartPs_SendByte
reeRTOS\xil_io\Xil_In32
eeRTOS\xil_io\Xil_Out32
S\xil_printf\xil_printf
reeRTOS\outbyte\outbyte
B::Trace.List /Track
Setup... Goto... Find... Chart Profile MIPS More Less

```
record  run address          cycle   data     symbol                              ti.back
            ...
            *
            * @return       None.
            *
            * @note         None.
            *
            ****************************************************************************
            void Xil_Out32(u32 OutAddress, u32 Value)
            {
   196                *(volatile u32 *) OutAddress = Value;
            str       r1,[r0]                ; Value,[r0]
            bx        r14
-001527         ZR:3F00080C ptrace             ..TOS\xil_printf\xil_printf+0x44     0.230us

   157          for ( ; *ctrl; ctrl++) {
            ldrb      r0,[r4,#0x1]
            add       r11,r4,#0x1            ; r11,ctrl,#1
            cmp       r0,#0x0                ; r0,#0
            beq       0x3F000A48

                      /* move format string chars to buffer until a  */
                      /* format control is found.                    */
   161                if (*ctrl != '%') {
            cmp       r0,#0x25               ; r0,#37
```

4.8 Summary view

In the pictures below, all the concepts previously discussed are shown as a summary global view of TRACE32 debugger, respectively for Zynq core 0 running a Linux kernel, and Zynq core 1 running a FreeRTOS based application.



```
B::term
root@bora:/lib/modules# insmod xilinx-xadcps.ko
root@bora:/lib/modules# ls -l /
total 60
drwxr-xr-x  2 ubuntu ubuntu 4096 Nov 20  2013 bin
drwxr-xr-x  2 ubuntu ubuntu 4096 Sep  5  2013 boot
drwxr-xr-x  6 root   root   3200 Jan  1 00:00 dev
drwxr-xr-x 42 ubuntu ubuntu 4096 Nov 21  2013 etc
drwxr-xr-x  3 ubuntu ubuntu 4096 Nov 18  2013 home
drwxr-xr-x 11 ubuntu ubuntu 4096 Nov 20  2013 lib
```

```
B::MMU.List PageTable
address                          physical
C:0000:00000000--00007FFF
C:0000:00008000--00008FFF        A:1CC64000--1CC64FFF
C:0000:00009000--00009FFF        A:1EE4E000--1EE4EFFF
C:0000:0000A000--0000AFFF        A:1CE5D000--1CE5DFFF
C:0000:0000B000--0000BFFF        A:1EDA3000--1EDA3FFF
C:0000:0000C000--0000CFFF        A:1EC27000--1EC27FFF
C:0000:0000D000--0000DFFF        A:1ED62000--1ED62FFF
C:0000:0000E000--0000EFFF        A:00BC2000--00BC2FFF
                                 A:1EE09000--1EE09FFF
                                 A:1EDE4000--1EDE4FFF
                                 A:00BCF000--00BCFFFF
                                 A:1CC9A000--1CC9AFFF
                                 A:1CCCE000--1CCCEFFF
                                 A:1C825000--1C825FFF
                                 A:00B6B000--00B6BFFF
```

```
B::TASK.Process
magic       command            #thr  state     spaceid  pids
C057F408  ⊞ swapper/0          30.   current   0000     0. 2. 3. 4.
DD845B80    init               -     sleeping  0001     1.
DD98F840    upstart-udev-br    -     sleeping  02CC     716.
DD98CBC0    udevd              -     sleeping  02CE     718.
DD91B400    udevd              -     sleeping  0308     776.
DD91B040    udevd              -     sleeping  0309     777.
DD92E440    upstart-socket-    -     sleeping  0482     1154.
DD98FC00    getty              -     sleeping  0489     1161.
DD9927C0    getty              -     sleeping  048B     1163.
DD927080    getty              -     sleeping  048D     1165.
DD929480    getty              -     sleeping  048E     1166.
DDA1FB80    getty              -     sleeping  0494     1172.
DD927440    getty              -     sleeping  04A8     1192.
DD98F480    login              -     sleeping  04A9     1193.
DD98A7C0    bash               -     sleeping  04B5     1205.
```

```
B::List
addr/line          code      label      mnemonic                comment
                   }
2966               asmlinkage void __sched schedule(void)
2967               {
ZSR:0000:C03C6FA4  E92D4800  schedule: push  {r11,r14}
                   ...
                   /*
                    * how to get the thread information struct from C
                    */
                   static inline struct thread_info *current_thread_in

                   static inline struct thread_info *current_thread_in
                   {
                           register unsigned long sp asm ("sp");
99                         return (struct thread_info *)(sp & ~(THREAD
ZSR:0000:C03C6FA8  E1A0100D            cpy     r1,r13           ; r1,s
ZSR:0000:C03C6FAC  E3C10D7F            bic     r0,r1,#0x1FC0    ; r0,r
2967               {
ZSR:0000:C03C6FB0  E28DB004            add     r11,r13,#0x4     ; r11,
                   ...
                   /*
                    * how to get the thread information struct from C
                    */
                   static inline struct thread_info *current_thread_in
```

```
B::AREA
Please re
Warning:
file 'C:\
Warning:
file 'C:\
file 'C:\
initializ
MMU scan
file 'C:\
mode: VT100, method: COM COM6, 115200 baud, 8 bits, no parity, 1 stop bits, no h
autoloader set.
autoloader set.
file 'C:\T32+\T32
file C:\T32+\T32-
```

```
B::Frame /Locals /Caller
-000  schedule()

-001  worker_thread(
        ⊞ __worker = 0xD
      ⊞ worker = 0xDD8AA
      ⊞ pool = 0xC09C250
      · __warned = FALSE
      · __warned = FALSE
      · __warned = FALSE
      · flags = 4
      · __warned = FALSE
      · __warned = FALSE
      · __warned = FALSE
      · __warned = FALSE
      · __warned = FALSE
      · __warned = FALSE

              spin_unlock_irq(&pool->lock);
              schedule();
-002  kthread(
                    = 0xDD86FEBC)
        0xDD86FEBC
        = 0xC0036CA8
        DD8AA9C0
        lags = 0, cpu = 3556983908, data =
```

```
B::TASK.MODule
magic     name          state  size   address   refcount  depends
BF0018A8  xilinx_xadcps Live   7605.  BF000000
```

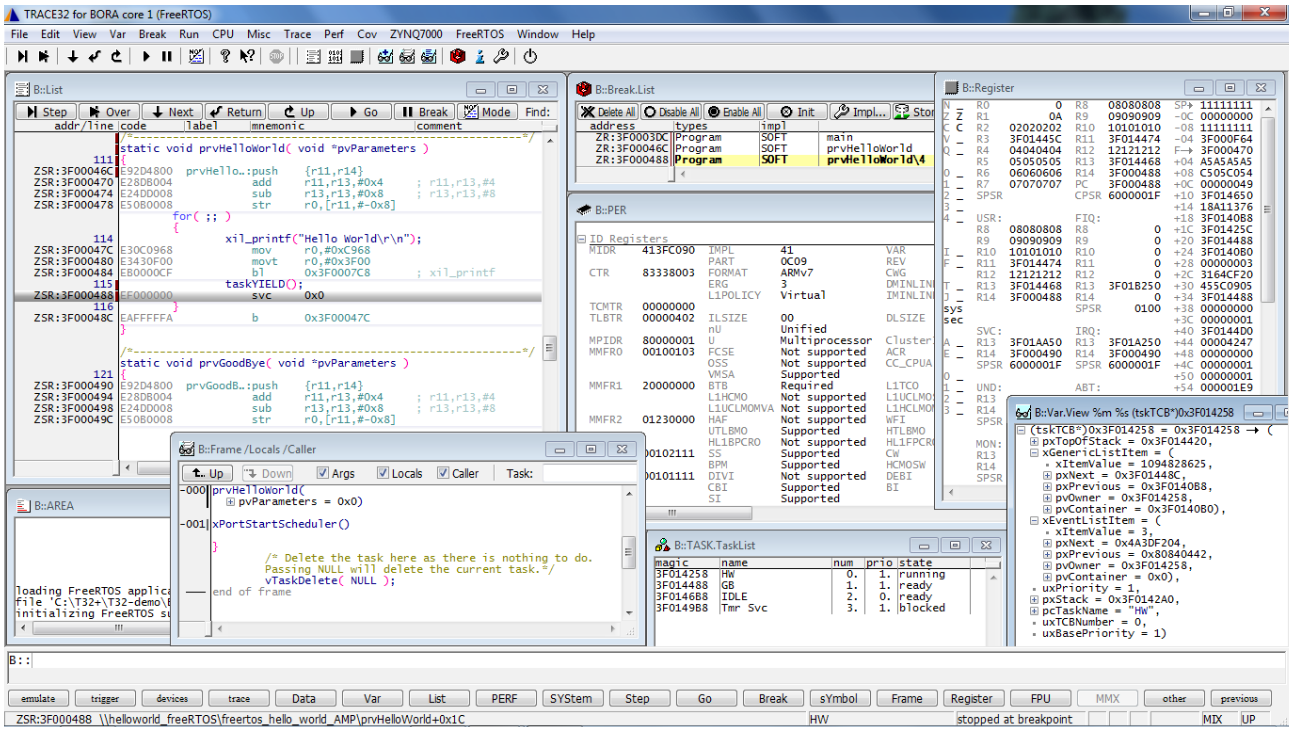
```
B::SYnch
synch: ( ) OFF  (●) ON   [RESet]
Connect: localhost:10000   localhost:10001
master: ☑ MasterGo  ☑ MasterBreak  ☑ MasterStep  ☑ MasterSystemMode
slave:  ☑ SlaveGo   ☑ SlaveBreak   ☑ SlaveStep   ☑ SlaveSystemMode
```

```
B::Break.List
address            types    impl
ZR:0000:C03C6FA4   Program  SOFT   schedule
```

```
B::
ZSR:0000:C03C6FA4  \\vmlinux\sched/core\schedule     kworker/0:1     stopped at breakpoint     MIX  UP
```

TRACE32 for BORA core 0 (Linux)
File Edit View Var Break Run CPU Misc Trace Perf Cov ZYNQ7000 Linux Window Help
Step Over Next Return Up Go Break Mode Find:
Up Down
Delete All Disable All Enable All Init Impl... Store... Load... Set...
emulate trigger devices trace Data Var List PERF SYStem Step Go Break sYmbol Frame Register FPU MMX other previous

Fig. 4: Linux at Zynq core 0


TRACE32 for BORA core 1 (FreeRTOS)
File Edit View Var Break Run CPU Misc Trace Perf Cov ZYNQ7000 FreeRTOS Window Help
B::List
Step Over Next Return Up Go Break Mode Find:
B::Break.List
Delete All Disable All Enable All Init Impl... Stor
B::PER
B::Register
B::AREA
B::Frame /Locals /Caller
Up Down Args Locals Caller Task:
B::TASK.TaskList
B::Var.View %m %s (tskTCB*)0x3F014258
emulate trigger devices trace Data Var List PERF SYStem Step Go Break sYmbol Frame Register FPU MMX other previous

```
   addr/line code     label       mnemonic             comment
                /*-----------------------------------------------------------*/
                static void prvHelloWorld( void *pvParameters )
          111   {
ZSR:3F00046C  E92D4800  prvHello..:push   {r11,r14}
ZSR:3F000470  E28DB004             add    r11,r13,#0x4       ; r11,r13,#4
ZSR:3F000474  E24DD008             sub    r13,r13,#0x8       ; r13,r13,#8
ZSR:3F000478  E50B0008             str    r0,[r11,#-0x8]
                        for( ;; )
                        {
          114                   xil_printf("Hello World\r\n");
ZSR:3F00047C  E30C0968             mov    r0,#0xC968
ZSR:3F000480  E3430F00             movt   r0,#0x3F00
ZSR:3F000484  EB0000CF             bl     0x3F0007C8         ; xil_printf
          115                   taskYIELD();
ZSR:3F000488  EF000000             svc    0x0
          116           }
ZSR:3F00048C  EAFFFFFA             b      0x3F00047C
                }

                /*-----------------------------------------------------------*/
                static void prvGoodBye( void *pvParameters )
          121   {
ZSR:3F000490  E92D4800  prvGoodB..:push   {r11,r14}
ZSR:3F000494  E28DB004             add    r11,r13,#0x4       ; r11,r13,#4
ZSR:3F000498  E24DD008             sub    r13,r13,#0x8       ; r13,r13,#8
ZSR:3F00049C  E50B0008             str    r0,[r11,#-0x8]
```

```
address        types     impl
ZR:3F0003DC    Program   SOFT    main
ZR:3F00046C    Program   SOFT    prvHelloWorld
ZR:3F000488    Program   SOFT    prvHelloWorld\4
```

```
ID Registers
MIDR     413FC090   IMPL          41               VAR
                    PART          0C09             REV
CTR      83338003   FORMAT        ARMv7            CWG
                    ERG           3                DMINLIN
                    L1POLICY      Virtual          IMINLIN
TCMTR    00000000
TLBTR    00000402   ILSIZE        00               DLSIZE
                    nU            Unified
MPIDR    80000001   U             Multiprocessor   Cluster
MMFR0    00100103   FCSE          Not supported    ACR
                    OSS           Not supported    CC_CPUA
                    VMSA          Supported
MMFR1    20000000   BTB           Required         L1TCO
                    L1HCMO        Not supported    L1UCLMO
                    L1UCLMOMVA    Not supported    L1HCLMO
MMFR2    01230000   HAF           Not supported    WFI
                    UTLBMO        Supported        HTLBMO
                    HL1BPCRO      Not supported    HL1FPCR
         00102111   SS            Supported        CW
                    BPM           Supported        HCMOSW
         00101111   DIVI          Not supported    DEBI
                    CBI           Supported        BI
                    SI            Supported
```

```
N _  R0           0   R8    08080808   SP> 11111111
Z Z  R1          0A   R9    09090909   -0C 00000000
C C  R2    02020202   R10   10101010   -08 11111111
V _  R3    3F01445C   R11   3F014474   -04 3F000F64
Q _  R4    04040404   R12   12121212   F-> 3F000470
     R5    05050505   R13   3F014468   +04 A5A5A5A5
0 _  R6    06060606   R14   3F000488   +08 C505C054
1 _  R7    07070707   PC    3F000488   +0C 00000049
2 _  SPSR             CPSR  6000001F   +10 3F014650
3 _                                    +14 18A11376
4 _  USR:             FIQ:             +18 3F0140B8
     R8    08080808   R8           0   +1C 3F01425C
     R9    09090909   R9           0   +20 3F014488
I _  R10   10101010   R10          0   +24 3F0140B0
F _  R11   3F014474   R11          0   +28 00000003
     R12   12121212   R12          0   +2C 3164CF20
T _  R13   3F014468   R13   3F01B250   +30 455C0905
J _  R14   3F000488   R14          0   +34 3F014488
sys                   SPSR      0100   +38 00000000
sec                                    +3C 00000001
     SVC:             IRQ:             +40 3F0144D0
A _  R13   3F01AA50   R13   3F01A250   +44 00004247
E _  R14   3F000490   R14   3F000490   +48 00000000
     SPSR  6000001F   SPSR  6000001F   +4C 00000001
0 _                                    +50 00000001
1 _  UND:             ABT:             +54 000001E9
2 _  R13
3 _  R14
     SPSR

     MON:
     R13
     R14
     SPSR
```

```
loading FreeRTOS applica
file 'C:\T32+\T32-demo\E
initializing FreeRTOS su
```

```
-000 prvHelloWorld(
       pvParameters = 0x0)

-001 xPortStartScheduler()

     }
             /* Delete the task here as there is nothing to do.
             Passing NULL will delete the current task.*/
             vTaskDelete( NULL );
---- end of frame
```

```
magic     name      num  prio  state
3F014258  HW         0.    1.  running
3F014488  GB         1.    1.  ready
3F0146B8  IDLE       2.    0.  ready
3F0149B8  Tmr Svc    3.    1.  blocked
```

```
(tskTCB*)0x3F014258 = 0x3F014258 -> (
  pxTopOfStack = 0x3F014420,
  xGenericListItem = (
    xItemValue = 1094828625,
    pxNext = 0x3F01448C,
    pxPrevious = 0x3F0140B8,
    pvOwner = 0x3F014258,
    pvContainer = 0x3F0140B0),
  xEventListItem = (
    xItemValue = 3,
    pxNext = 0x4A3DF204,
    pxPrevious = 0x80840442,
    pvOwner = 0x3F014258,
    pvContainer = 0x0),
  uxPriority = 1,
  pxStack = 0x3F0142A0,
  pcTaskName = "HW",
  uxTCBNumber = 0,
  uxBasePriority = 1)
```

```
B::
ZSR:3F000488  \\helloworld_freeRTOS\freertos_hello_world_AMP\prvHelloWorld+0x1C     HW     stopped at breakpoint     MIX  UP
```

Fig. 5: FreeRTOS at Zynq core 1

4.9 Lauterbach References

This chapter has been written by Lauterbach italian branch office.

Contact information:

Lauterbach SRL

Via Enzo Ferrieri 12

20153 Milan (Italy)

Tel. +39 02 45490282

Email info_it@lauterbach.it

Web www.lauterbach.it

5 Appendixes

5.1 U-Boot environment

The following is the U-Boot environment that can be printed using the `print` command (please note these are examples and that the actual variables may differ for different U-Boot settings), for both the FreeRTOS applications described in this application note.

5.1.1 Example #1: HelloWorld application

```
addcons=setenv bootargs ${bootargs} console=${console},115200n8 cma=16M debug maxcpus=$
{nr_cpus}
addip=setenv bootargs ${bootargs} ip=${ipaddr}:${serverip}:${gateway}:${netmask}:$
{hostname}:${ethdev}
addmem=setenv bootargs ${bootargs} mem=1008M
baudrate=115200
bootdelay=3
bootfile=bora/uImage
console=ttyPS0
dt_base=0x005C0000
ethact=Gem.e000b000
ethaddr=00:0a:35:00:01:22
ethdev=eth0
fdt_high=0x1F000000
fileaddr=0
filesize=2f0ed0
fpga_base=0x00180000
fpga_file=belk/free_rtos/bora_design_wrapper.bin
freertos_addr=0x3F000000
freertos_file=bora/belk/free_rtos/helloworld_freertos.bin
fsbl_base=0x40000
ftd_file=bora/bora.dtb
gateway=192.168.0.254
header_base=0
hostname=bora
ipaddr=192.168.0.209
jtagboot=echo TFTPing Linux to RAM...;tftp 0x8000 zImage;tftp 0x1000000 devicetree.dtb;tftp
0x800000 ramdisk8M.image.gz;go 0x8000
kernel_size=0x140000
load=tftp ${loadaddr} bora/u-boot.bin
load_dt=tftp ${loadaddr} bora/bora.dtb
load_fpga=tftp ${loadaddr} bora/${fpga_file}
load_freertos=tftp ${freertos_addr} ${freertos_file};mw.l 0xFFFFFFF0 ${freertos_addr}
load_fsbl=tftp ${loadaddr} bora/belk/free_rtos/bora_FSBL.bin
load_h=tftp ${loadaddr} bora/boot_header-1.1.0
loadaddr=0x08000000
loadaddr_ftd=0x01000000
loadaddr_kern=0x0
loadk=tftp ${loadaddr_kern} ${bootfile};tftp ${loadaddr_ftd} ${ftd_file}
modeboot=qspiboot
net_nfs=run program_fpga; run load_freertos; run loadk nfsargs addip addcons addmem; bootm $
{loadaddr_kern} - ${loadaddr_ftd}
netmask=255.255.255.0
nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=${serverip}:${rootpath}
nr_cpus=1
program_fpga=run load_fpga;fpga load 0 ${loadaddr} 0x${filesize}
qspiboot=sf probe 0 0 0;sf read 0x8000 0x100000 0x2c0000;sf read 0x1000000 0x3c0000
0x40000;sf read 0x800000 0x400000 0x800000;go 0x8000
```

```
ramdisk_size=0x200000
rootpath=/home/shared/devel/dave/bora-DBRx/sw/linux/rfs/BELK/ubuntu_12.04
sdboot=echo Copying Linux from SD to RAM...;mmcinfo;fatload mmc 0 0x8000 zImage;fatload mmc
0 0x1000000 devicetree.dtb;fatload mmc 0 0x800000 ramdisk8M.image.gz;go 0x8000
serverip=192.168.0.23
stderr=serial
stdin=serial
stdout=serial
u-boot_base=0x80000
update=sf probe 0 0 0;sf erase ${u-boot_base} 0x80000;sf write ${loadaddr} ${u-boot_base}
0x80000
update_dt=sf probe 0 0 0;sf erase ${dt_base} 0x40000;sf write ${loadaddr} ${dt_base} 0x40000
update_fpga=sf probe 0 0 0;sf erase ${fpga_base} 0x440000;sf write ${loadaddr} ${fpga_base}
0x440000
update_fsbl=sf probe 0 0 0;sf erase ${fsbl_base} 0x40000;sf write ${loadaddr} ${fsbl_base}
0x40000
update_fsbl_nand=run load_fsbl; nand erase 0x40000 0x40000; nand write ${loadaddr} 0x40000
0x40000
update_h=sf probe 0 0 0;sf erase ${header_base} 0x40000;sf write ${loadaddr} ${header_base}
0x40000
```

5.1.2 Example #2: RPMsg-based application

```
addcons=setenv bootargs ${bootargs} console=${console},115200n8 cma=16M debug
addip=setenv bootargs ${bootargs} ip=${ipaddr}:${serverip}:${gateway}:${netmask}:$
{hostname}:${ethdev}
addmem=setenv bootargs ${bootargs} mem=496M
baudrate=115200
bootdelay=3
bootfile=bora/uImage
console=ttyPS0
dt_base=0x005C0000
ethact=Gem.e000b000
ethaddr=00:0a:35:00:01:22
ethdev=eth0
fdt_high=0x1F000000
fileaddr=0
filesize=2f0ed0
fpga_base=0x00180000
fpga_file=belk/free_rtos/bora_design_wrapper.bin
freertos_addr=0x3F000000
freertos_file=bora/belk/free_rtos/helloworld_freertos.bin
fsbl_base=0x40000
ftd_file=bora/bora.dtb
gateway=192.168.0.254
header_base=0
hostname=bora
ipaddr=192.168.0.209
jtagboot=echo TFTPing Linux to RAM...;tftp 0x8000 zImage;tftp 0x1000000 devicetree.dtb;tftp
0x800000 ramdisk8M.image.gz;go 0x8000
kernel_size=0x140000
load=tftp ${loadaddr} bora/u-boot.bin
load_dt=tftp ${loadaddr} bora/bora.dtb
load_fpga=tftp ${loadaddr} bora/${fpga_file}
load_freertos=tftp ${freertos_addr} ${freertos_file};mw.l 0xFFFFFFF0 ${freertos_addr}
load_fsbl=tftp ${loadaddr} bora/belk/free_rtos/bora_FSBL.bin
load_h=tftp ${loadaddr} bora/boot_header-1.1.0
loadaddr=0x08000000
loadaddr_ftd=0x01000000
loadaddr_kern=0x0
loadk=tftp ${loadaddr_kern} ${bootfile};tftp ${loadaddr_ftd} ${ftd_file}
modeboot=qspiboot
net_nfs=run program_fpga; run loadk nfsargs addip addcons addmem; bootm ${loadaddr_kern} - $
{loadaddr_ftd}
netmask=255.255.255.0
nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=${serverip}:${rootpath}
nr_cpus=1
program_fpga=run load_fpga;fpga load 0 ${loadaddr} 0x${filesize}
qspiboot=sf probe 0 0 0;sf read 0x8000 0x100000 0x2c0000;sf read 0x1000000 0x3c0000
0x40000;sf read 0x800000 0x400000 0x800000;go 0x8000
ramdisk_size=0x200000
```

```
rootpath=/home/shared/devel/dave/bora-DBRx/sw/linux/rfs/BELK/ubuntu_12.04
sdboot=echo Copying Linux from SD to RAM...;mmcinfo;fatload mmc 0 0x8000 zImage;fatload mmc
0 0x1000000 devicetree.dtb;fatload mmc 0 0x800000 ramdisk8M.image.gz;go 0x8000
serverip=192.168.0.23
stderr=serial
stdin=serial
stdout=serial
u-boot_base=0x80000
update=sf probe 0 0 0;sf erase ${u-boot_base} 0x80000;sf write ${loadaddr} ${u-boot_base}
0x80000
update_dt=sf probe 0 0 0;sf erase ${dt_base} 0x40000;sf write ${loadaddr} ${dt_base} 0x40000
update_fpga=sf probe 0 0 0;sf erase ${fpga_base} 0x440000;sf write ${loadaddr} ${fpga_base}
0x440000
update_fsbl=sf probe 0 0 0;sf erase ${fsbl_base} 0x40000;sf write ${loadaddr} ${fsbl_base}
0x40000
update_fsbl_nand=run load_fsbl; nand erase 0x40000 0x40000; nand write ${loadaddr} 0x40000
0x40000
update_h=sf probe 0 0 0;sf erase ${header_base} 0x40000;sf write ${loadaddr} ${header_base}
0x40000
```