BORA

ARM Cortex-A9 + FPGA CPU Module

Ultra Line

Bora Embedded Linux Kit (*BELK*)

AN-BELK-002

Trace on the Bora AMP (Linux + FreeRTOS) system

DAVE Embedded Systems www.dave.eu info@dave.eu

<Page intentionally left blank>

Table of Contents

Illustration Index

1 Preface

1.1 About this document

This application note describes how to configure TRACE32 ® debugger to support debug and trace of Linux running on the first Zynq core, and FreeRTOS, running on the second Zynq core. This application note is the proper continuation of the previously releseased AN-BELK-001 “AMP on Bora - Linux and FreeRTOS”, which describes how to build the software components required to set up asymmetric multi-processing (AMP)” on Bora.

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
AN-BELK-001 “AMP on Bora - Linux and FreeRTOS”	Provided with BELK

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	May 2014	First release

2 Introduction

2.1 Bora SOM

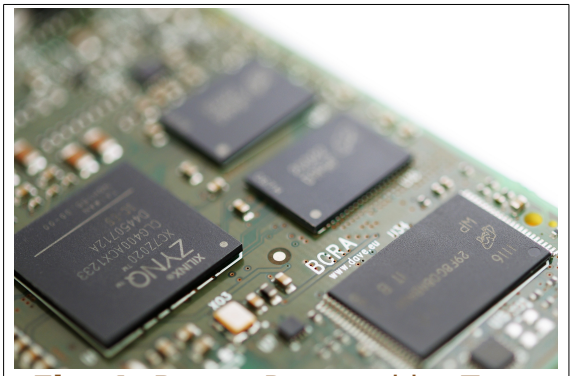

Fig. 1: Bora – Powered by Zynq processor

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.

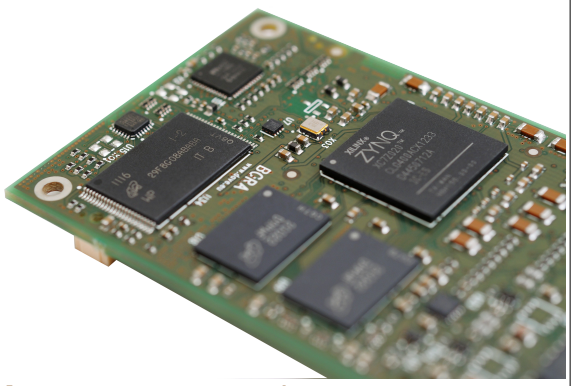

Fig. 2: Bora – Dual ARM Cortex A9 plus FPGA

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.

2.5 Trace

Embedded Trace Macrocells™ (ETM) provide comprehensive debug and trace facilities for ARM processors. They allow capture of information on the processor both before and after a specific event, while adding no burden to the processor's performance, while the processor runs at full speed. A software debugger provides the user interface to the ETM and configures all the ETM facilities, such as the trace port, typically using a JTAG interface. The debugger also displays the trace information that has been captured.

The ETM compresses the trace information and either:

Exports it through a trace port. An external Trace Port Analyzer (TPA) captures the trace information.

Writes it directly to an on-chip Embedded Trace Buffer (ETB). The trace is read out at low speed using the JTAG interface when the trace capture is complete.

When the trace has been captured the debugger extracts the information from the TPA or ETB and decompresses it to provide a full disassembly, with symbols, of the code that was executed. The debugger can also link this back to the original high-level source code, providing you with a visualization of how the code was executed on the target system.

2.6 AMP on Bora (AN-BELK-001)

The application note **AN-BELK-001 "AMP on Bora - Linux and FreeRTOS"** describes how to build the software components required to set up asymmetric multi-processing (AMP) configuration required to run Linux OS on firstCortex-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. The application note describes also some advanced debugging techniques for AMP configuration, introducing the Lauterbach TRACE32 ® debugger setup for multi-core environments. Reading of AN-BELK-001 is a prerequisite for using the techniques described in this document.

3 Tracing on Bora AMP

3.1 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.

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" of Zynq Reference Manual for more details).

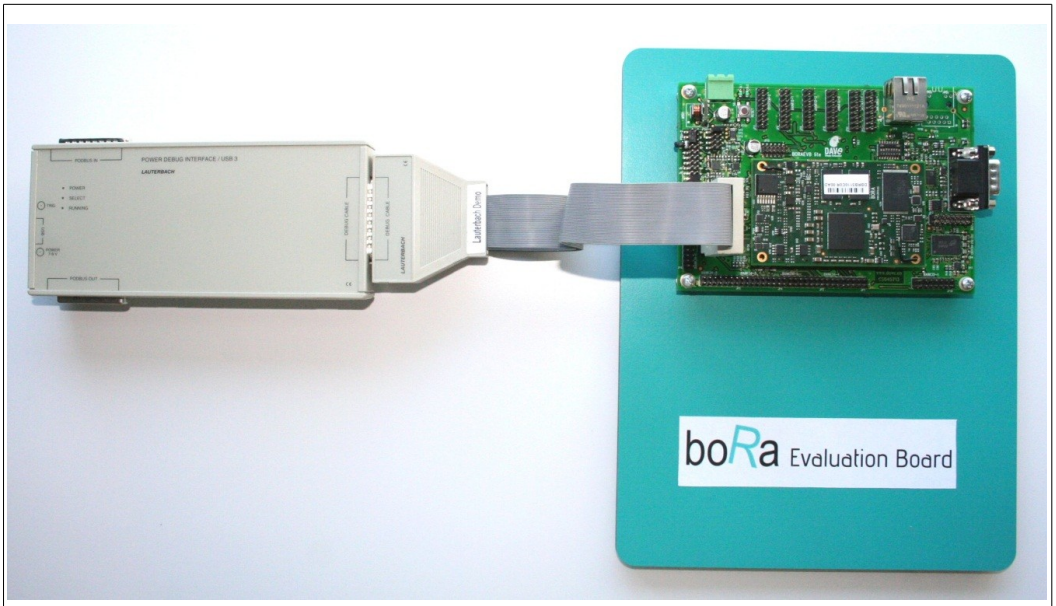

Fig. 3: BoraEVB connected to Lauterbach PowerDebug Interface/USB3

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

3.2 Prerequisites

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)

For detailed information on how to setup the Bora AMP system, please refer to **AN-BELK-001 "AMP on Bora - Linux and FreeRTOS"**.

3.2.1 Debug and onchip trace

- 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)

3.2.2 Debug and offchip trace

- LA-7690 PowerTrace Ethernet or LA-7699 PowerDebug II + LA-7692 PowerTrace II 1 GigaByte
- LA-7843 JTAG Debugger for Cortex-A/-R
- LA-7960X License for Multicore Debugging
- LA-7992 Preproc. for ARM-ETM/AUTOFOCUS II 600 Flex or LA-7993 Preproc. for ARM-ETM/AUTOFOCUS 600 MIPI
- TRACE32 PowerView for ARM (Release ≥ Feb 2013, Software Version: R.2013.02.000045901)

3.3 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.

For detailed information on how to setup the TRACE32 configuration, please refer to Section 4 of the AN-BELK-001 “AMP on Bora - Linux and FreeRTOS” application note.

3.4 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

&core1 Synch.XTRACK localhost:&intercomport_core0
```

Each trace window will follow the time / record synchronization of other trace windows, if the option /Track is used.

The following images show Linux onchip trace XTRACKED vs FreeRTOS onchip trace:

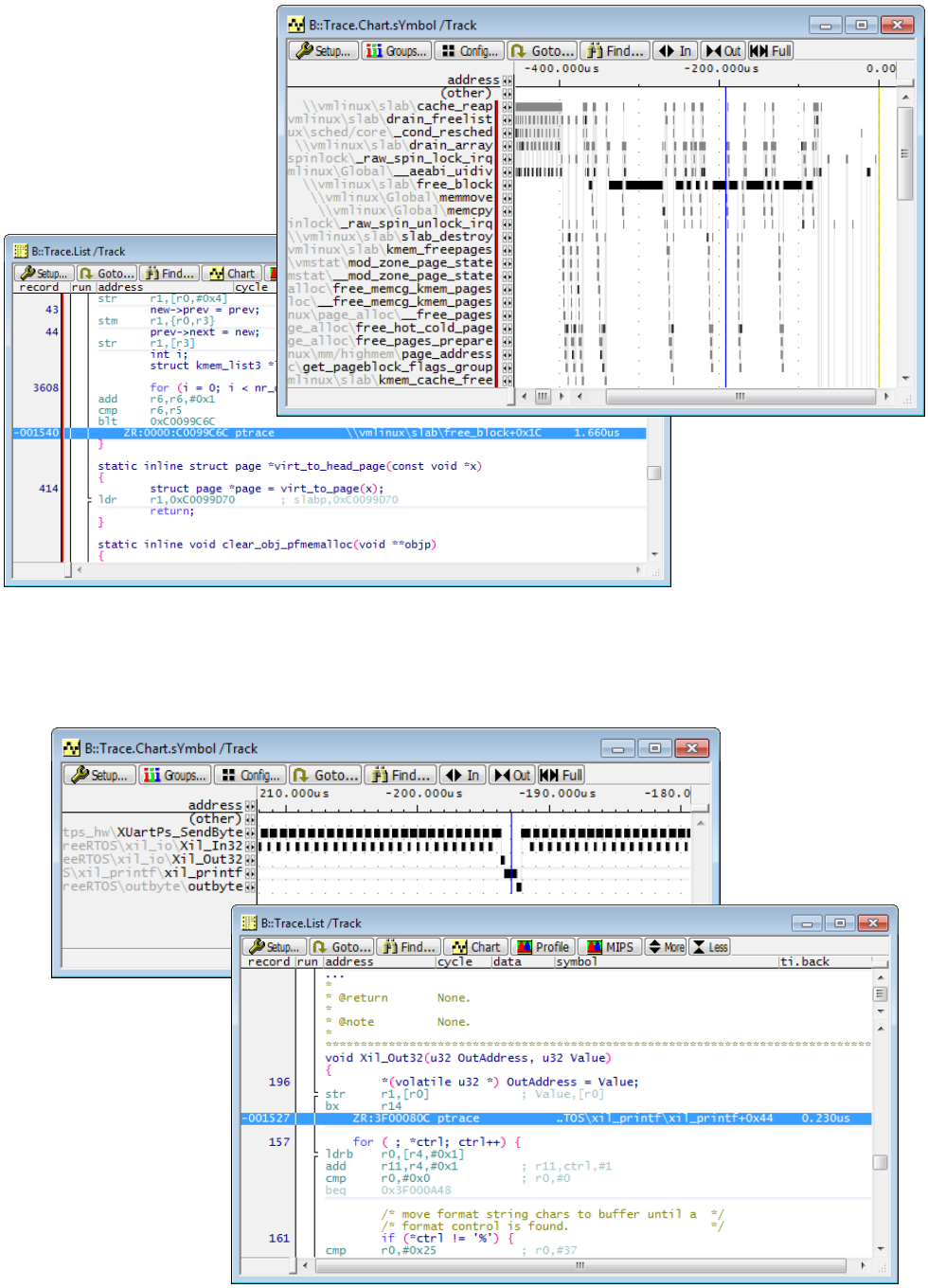

Fig. 4: Linux and FreeRTOS onchip trace

For further details, please refer to “Training Real-time Trace” manual (training_trace.pdf).

3.5 Offchip trace

The chip Zynq implements the ETM (Embedded Trace Macrocell), a CoreSight hardware component providing trace functionality. The data can be combined with other trace sources or passed directly off-chip to the Trace Port Interface Unit (TPIU). There it will be captured by a trace port analyzer (ETM Preprocessor).

3.5.1 Software prerequisites

Before using trace features, the Bora SOM must be properly programmed. Dedicated FPGA bistream and FSBL must be used, and they must be built following the instructions reported in sections 3.2.1 and 3.2.2 of the AN-BELK-001 “AMP on Bora - Linux and FreeRTOS”. Please note that the branch **bora-dev-trace** from the Bora git repository must be used.

3.5.2 Trace operations

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

```
Trace.METHOD Analyzer ; select the ETM+TPIU 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

&core1 Synch.XTRACK localhost:&intercomport_core0
```

Each trace window will follow the time / record synchronization

of other trace windows, if the option /Track is used. The command Trace.AutoFocus configures a preprocessor for an error-free sampling on a high-speed trace port.

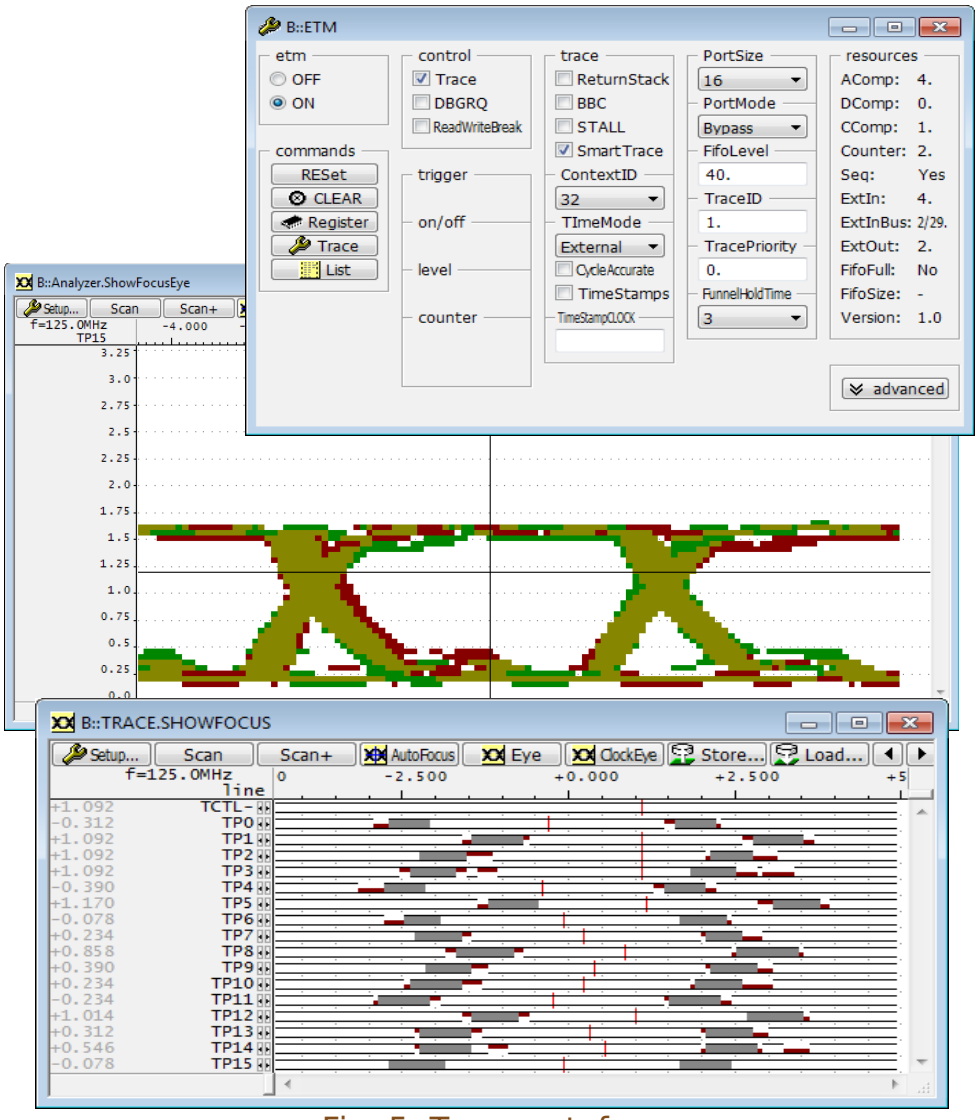

Fig. 5: Trace autofocus

For more details, please refer to “ARM-ETM Training” manual (training_arm_etm.pdf).

3.5.3 Linux offchip trace XTRACKED vs FreeRTOS offchip trace

The following images show Linux offchip trace XTRACKED vs FreeRTOS offchip trace:

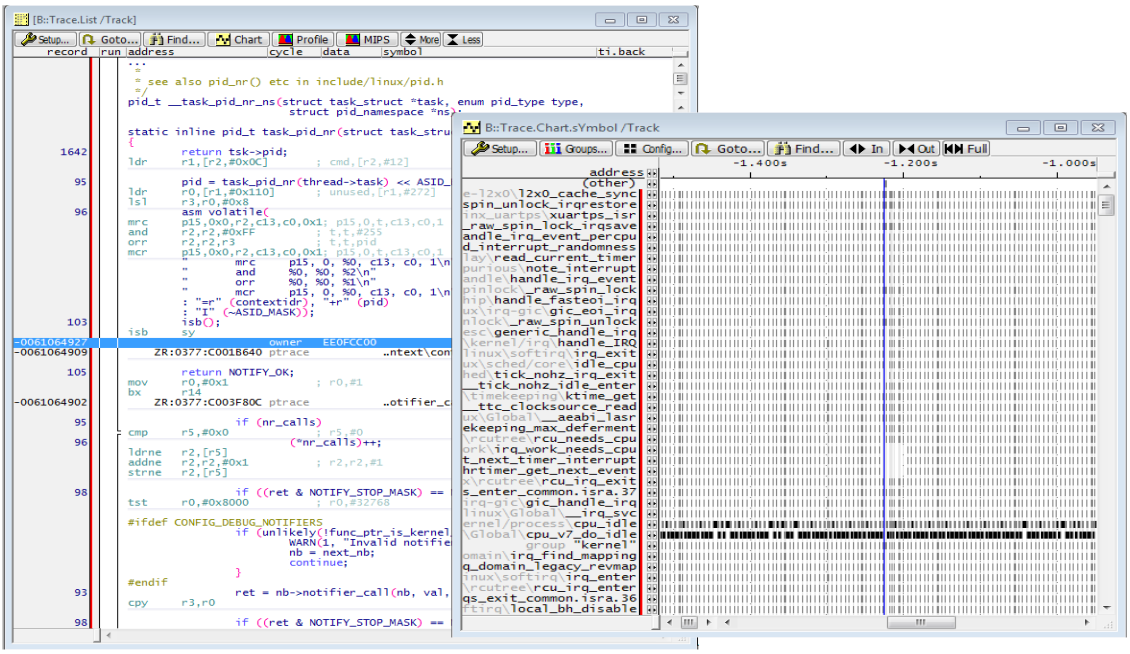

Fig. 6: Linux offchip trace

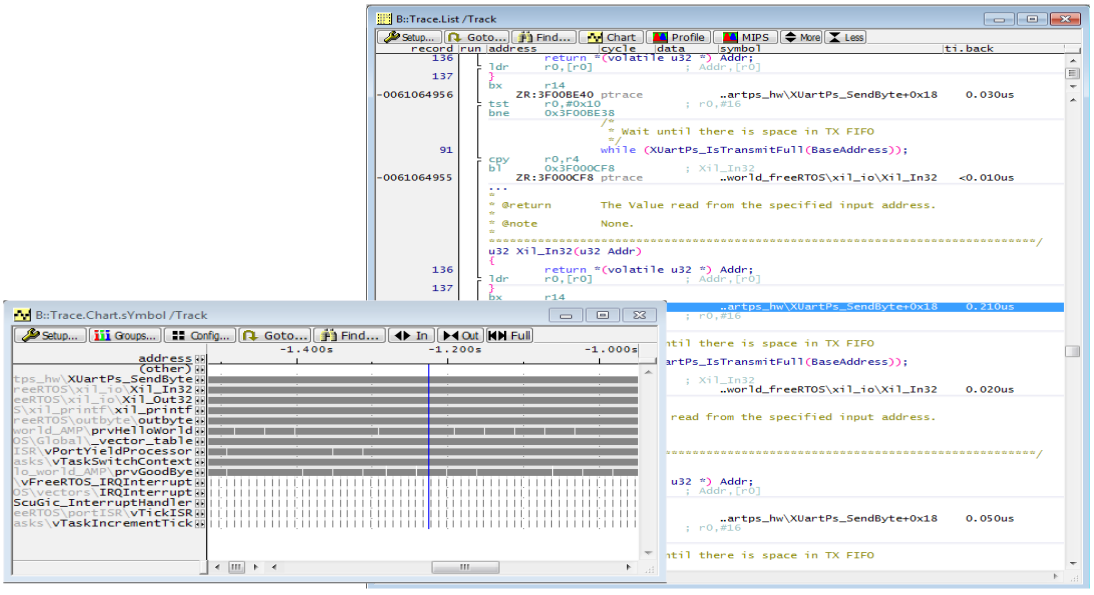

Fig. 7: FreeRTOS offchip trace

3.6 Summary view

In the pictures below, all the concepts discussed in both this application note and AN-BELK-001 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.

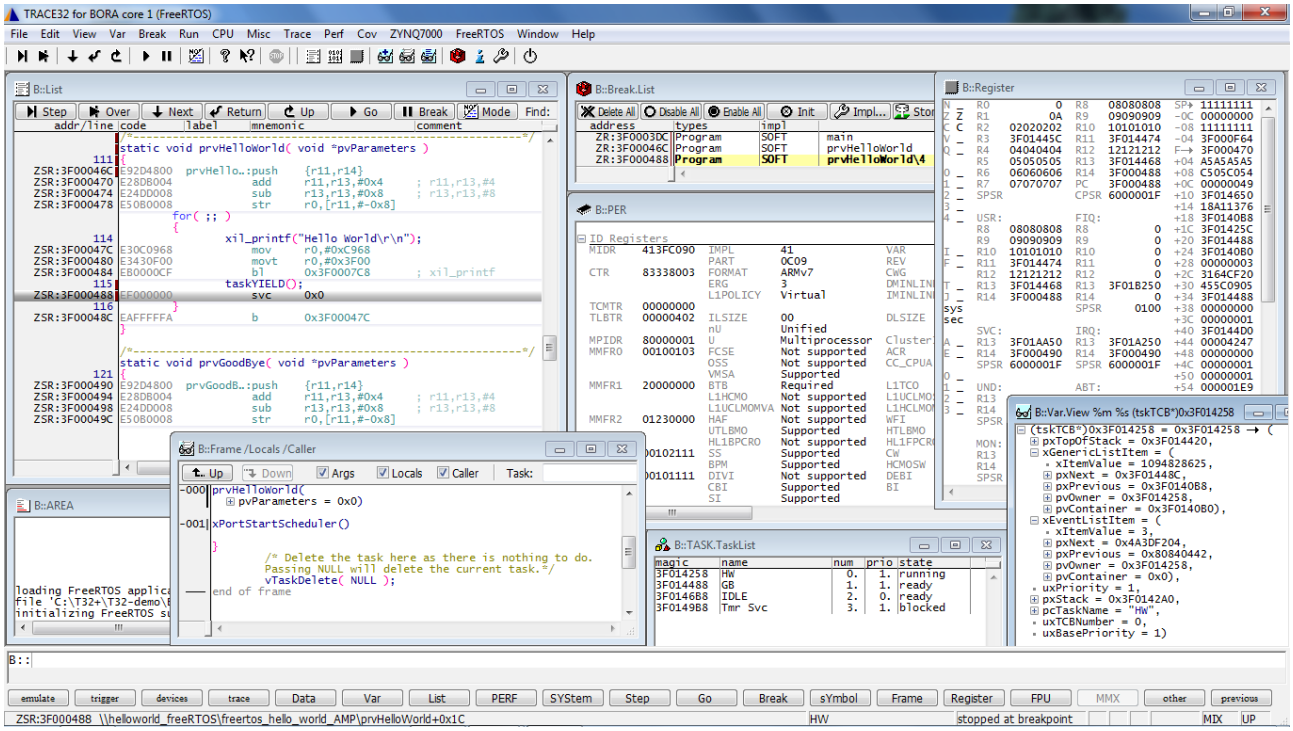

Fig. 8: FreeRTOS at Zynq core 1

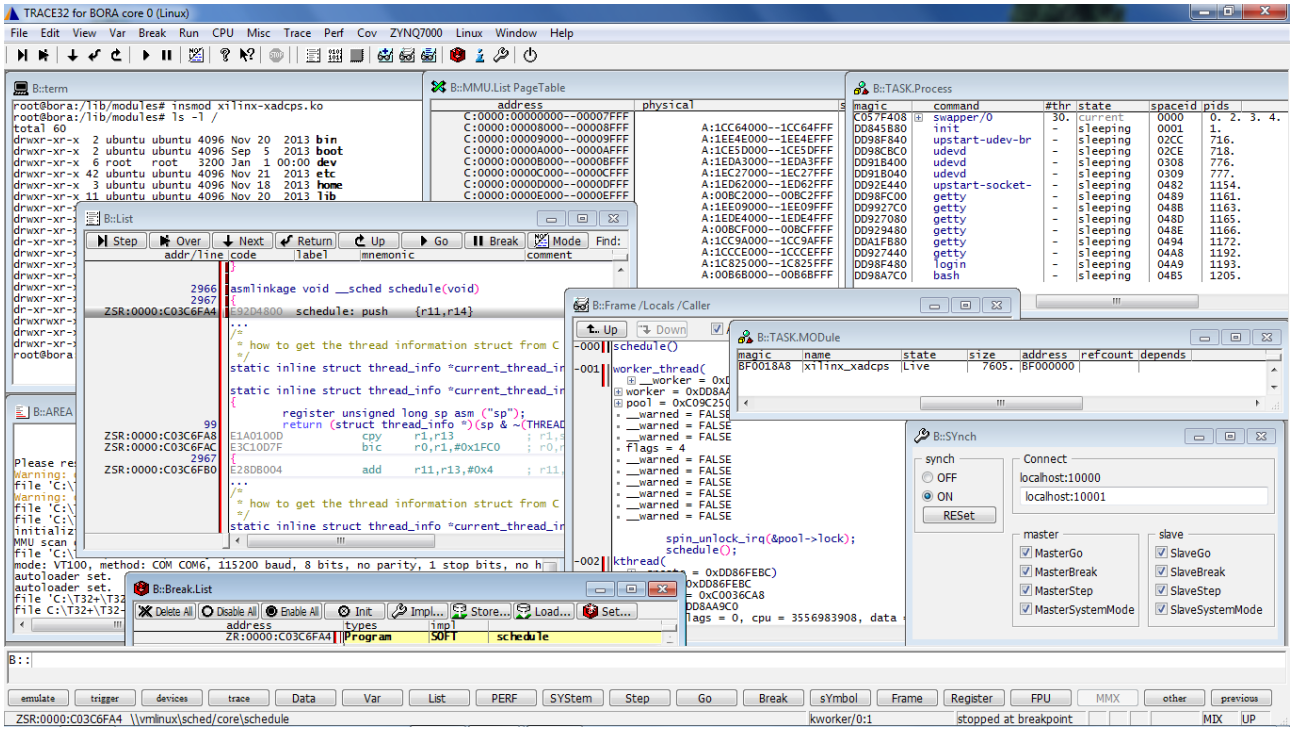

Fig. 9: Linux at Zynq core 0

3.7 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

4 Appendixes

The following sections report the scripts used in this application note and in AN-BELK-001.

4.1 amp_start_core0.bat

```
REM script to start debugger for core_0.
REM other core(s) will be started by practice script

SET t32path=c:\t32\bin\windows64
IF NOT EXIST %t32path%\t32marm.exe SET t32path=c:\t32\bin\windows
IF NOT EXIST %t32path%\t32marm.exe SET t32path=c:\t32\

if "%1" == "" goto useusb

start %t32path%\t32marm -c amp_config.t32,amp_demo.cmm 10000 TRACE32_A NET
NODE=%1 PACKLEN=1024 CORE=1
goto:eof

:useusb
start %t32path%\t32marm -c amp_config.t32,amp_demo.cmm 10000 TRACE32_A USB
CORE=1
```

4.2 amp_config.t32

```
; Trace32 Configuration file for multicore debugging
;
; Parameters:
;  t32marm -c amp_config.t32[,startscript] <intercom_port> <title>
; <interface to debug module> [interface options]
; Examples:
;  t32marm -c amp_config.t32,dualcoredemo.cmm 10000 TRACE32_A NET
; NODE=pod-rei-s1 PACKLEN=1024 CORE=1
;  t32marm -c amp_config.t32,dualcoredemo.cmm 10000 TRACE32_A USB CORE=1
;

IC=NETASSIST
PORT=${1}

; Environment variables
OS=
ID=T32${1}
TMP=C:\temp
SYS=C:\t32
HELP=c:\t32\pdf
```

```
PBI=
${3}
${4}
${5}
${6}

; Printer settings
PRINTER=WINDOWS

; Screen fonts
SCREEN=
FONT=SMALL
HEADER=Trace32 ${2}
```

4.3 amp_demo.cmm

```
;AMP dual-core debugging sample script for Zynq7000 based BORA target
;AMP: one Trace32 instance per core
;
; Remark:
; Linux run on CORE #1
; FreeRTOS application run on CORE #2 @ 0x3F000000
;

;define reference COM port for TERM
LOCAL &comPort
&comPort="COM6"

;define directory path for symbol files
LOCAL &freertosBasePath
LOCAL &uimageBasePath
LOCAL &dtbBasePath
LOCAL &vmlinuxBasePath

&freertosBasePath="./bora-build-20131029-nobk/bora.sdk/SDK/SDK_Export/hell
oworld_freeRTOS/Debug"
&uimageBasePath="."
&dtbBasePath="."
&vmlinuxBasePath="."

;define full name for symbol files
LOCAL &FreeRTOS_Bin
LOCAL &FreeRTOS_Elf
LOCAL &Linux_Bin
LOCAL &Linux_Elf

&FreeRTOS_Bin="&freertosBasePath/helloworld_freeRTOS.bin"
&FreeRTOS_Elf="&freertosBasePath/helloworld_freeRTOS.elf"
&Linux_Bin="&uimageBasePath/uImage"
```

```
&Linux_Elf="&vmlinuxBasePath/vmlinux"
&dtb="&dtbBasePath/bora.dtb"

;start Trace32 instance for core_1 (started only on demand)
  WAIT 1.s
  DO amp_demo_start_core1.cmm

;set up macros for intercom communication between the Trace32 instances
  &intercomport_core0=FORMAT.DECIMAL(1.,intercom.port())
  &intercomport_core1=FORMAT.DECIMAL(1.,intercom.port()+1)

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

;set titles
  &core0 Title "TRACE32 for BORA core 0 (Linux)"
  &core1 Title "TRACE32 for BORA core 1 (FreeRTOS)"

;arrange debugger windows
  &core0 FramePOS 0.0 0.0 152. 41.
  &core1 FramePOS 6.0 4.0 152. 41.
  &both  WinPAGE.RESet
  &both  WinCLEAR

;set working path of core_1 to match working path of core_0
  &path=OS.PWD()
  &core1 cd "&path"

;configure TERMinal inside TRACE32
  &core0 TERM.METHOD com &comPort 115200. 8 none 1stop none
  &core0 TERM.Mode vt100
  &core0 TERM.SIZE 80. 2000.
  &core0 TERM.SCROLL on
  &core0 WinPOS 0.5 0.0 80. 25. 0. 0. W000
  &core0 TERM

;debugger setup
  &both SCREEN.ALways
  &both MENU.RESet
  &both do ~~/t32
  &both do toolbar_quit_all.cmm
  &both SYNCH.RESet
  &both Break.RESet
  &both SYStem.RESet
  &both SYStem.Option EnReset OFF
  &both SYStem.JtagClock 10.0MHz

;set processor
  SYStem.CPU ZYNQ-7000CORE0
  SYStem.CONFIG.CORE 1 1
```

```
  SYStem.CONFIG.CTI.Base 0x80098000
  SYStem.CONFIG.CTI.Config CORTEXV1

  &core1 SYStem.CPU ZYNQ-7000CORE1
  &core1 SYStem.CONFIG.CORE 2 1
  &core1 SYStem.CONFIG.SLAVE ON
  &core1 SYStem.CONFIG.CTI.Base 0x80099000
  &core1 SYStem.CONFIG.CTI.Config CORTEXV1

;This selects the DAP for accessing the ARM cores
;(accessing the TAP of the FPGA logic requires different settings)
  &both SYStem.CONFIG DAPIRPRE 6.
  &both SYStem.CONFIG DAPIRPOST 0.
  &both SYStem.CONFIG DAPDRPRE 1.
  &both SYStem.CONFIG DAPDRPOST 0.

;check target power state
  &both AREA.CLEAR
  &both WinPOS 0.71429 30.0 80. 5. 0. 0. W001
  &both AREA

  IF STATE.POWER()
  (
    print "Please reset target now."
    ON RESET GOTO start
    STOP
  )
  ELSE
  (
    print "Please powerup target now."
    ON POWERUP GOTO start
    STOP
  )

start:
;reset handlers
  ON RESET
  ON POWERUP
  WAIT 1.s
;stop autoboot
  TERM.Out 13. 13.
  WAIT 100.ms

;core_0: halt on reset
  &core0 SYStem.Up
;core_1: still in reset, attach only
  &core1 SYStem.Mode.Attach
  &core1 Break

;debug Linux configuration for core 0
```

```
  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!
  SYStem.Option DACR ON                ; give Debugger global write
permissions

  ; load linux kernel binary, dtb and symbols
  Data.LOAD.Binary &Linux_Bin 0x00000000 /NoClear
  Data.LOAD.Binary &dtb 0x01000000 /NoClear
  sYmbol.RESet
  sYmbol.SourcePATH.Set .
  Data.LOAD.Elf &Linux_Elf /gnu /NoCODE /NoClear /SPART 5

  ; to remove automatic break when the boot process crosses Reset vector
  TrOnchip.Set RESET OFF

  b 0x00008000 /Onchip
  Go
  TERM.Out "setenv ipaddr 192.168.1.209" 13.
  wait 500.ms
  TERM.Out "setenv serverip 192.168.1.11" 13.
  wait 500.ms
  TERM.Out "setenv gateway 192.168.1.1" 13.
  wait 500.ms
  TERM.Out "setenv rootpath /opt/filesystemBora" 13.
  wait 500.ms
  TERM.Out "run nfsargs addcons addip" 13.
  wait 500.ms
  TERM.Out "bootm 0 - 0x01000000" 13.
  wait 500.ms
  wait !run()

  WinPOS 62.75 5.2857 80. 25. 22. 1. W002
  List
  DIALOG.OK "Linux is starting." "Press OK to continue."
  WinCLEAR W002

  SYStem.Option MMUspaces ON
  MMU.FORMAT LINUX swapper_pg_dir 0xc0000000--0xc1ffffff 0x00000000
  TRANSlation.Create 0xc0000000--0xc1ffffff 0x00000000
  TRANSlation.COMMON 0xc0000000--0xffffffff
  TRANSlation.TableWalk ON
  TRANSlation.ON

  print "initializing Linux support..."
  TASK.CONFIG ~~/demo/arm/kernel/linux/linux-3.x/linux3.t32       ; loads
Linux awareness
  MENU.ReProgram ~~/demo/arm/kernel/linux/linux-3.x/linux.men     ; loads
Linux menu
  HELP.FILTER.Add rtoslinux                                       ; add
linux awareness manual to help filter
```

```
  TASK.sYmbol.Option MMUSCAN OFF  ; not necessary with tablewalk

  ; switch on symbol autoloader
  ; not necessary starting from TRACE32 DVD February 2013
  ; sYmbol.AutoLoad.CHECKLINUX "do
~~/demo/arm/kernel/linux/linux-3.x/autoload "

  ; Group kernel area to be displayed with red bar
  GROUP.Create "kernel" 0xc0000000--0xffffffff /RED

  ; skip mmu initialization
  b schedule /Onchip
  Go
  wait !run()
  WinPOS 62.75 5.2857 80. 25. 22. 1. W002
  List
  ; let Linux run up to login prompt
  Go
  WAIT 10.s
  Break

;debug FreeRTOS configuration for core 1

  ; load FreeRTOS application symbols
  &core0 Data.LOAD.Binary &FreeRTOS_Bin a:0x3F000000 /Verify /NoClear
  &core1 print "loading FreeRTOS application symbols...."
  &core1 sYmbol.RESet
  &core1 sYmbol.SourcePATH.Set .
  &core1 Data.LOAD.Elf &FreeRTOS_Elf /NoCODE /SPART 6
  ; set core#2 application pointer
  &core0 D.S AZSD:0xFFFFFFF0 %LE %LONG 0x3F000000
  ; set FreeRTOS breakpoint
;  &core1 B.S 0x3F000000 /Onchip
;  &core1 B.S main /Onchip
;  &core1 B.S prvGoodBye /Program /SOFT
   &core1 B.S prvHelloWorld /Program /SOFT

  ; now you can debug FreeRTOS tasks
  &core1 Go
  &core1 wait !run()
  &core1 WinPOS 0.57143 0.16667 80. 23. 16. 1. W002
  &core1 List

  ; initialize RTOS support
  &core1 print "initializing FreeRTOS support..."
  &core1 TASK.CONFIG ~~/demo/arm/kernel/freertos/freertos.t32      ; load
FreeRTOS awareness (freertos.t32)
  &core1 MENU.ReProgram ~~/demo/arm/kernel/freertos/freertos.men   ; load
FreeRTOS menu (freertos.men)
  &core1 HELP.FILTER.Add rtosfreertos                              ; add
```

```
FreeRTOS awareness manual to filtered help

;setup for synchronous step/go/break and system mode change
  &core0 SYNCH.CONNECT localhost:&intercomport_core1
  &core1 SYNCH.CONNECT localhost:&intercomport_core0

  &both SYNCH.MasterGO         ON
  &both SYNCH.SlaveGO          ON
  &both SYNCH.MasterBREAK      ON
  &both SYNCH.SlaveBREAK       ON
  &both SYNCH.MasterSTEP       ON
  &both SYNCH.SlaveSTEP        ON
  &both SYNCH.MasterSystemMode ON
  &both SYNCH.SlaveSystemMode  ON

  &core0 TargetSystem DEFault Title SYnch.All InterComPort /Global

  &core0 Synch.XTRACK localhost:&intercomport_core1
  &core1 Synch.XTRACK localhost:&intercomport_core0

;trace configuration
; &both  Trace.METHOD Onchip
; or:

; &both  Trace.METHOD Analyzer
; &both  ETM.PORTSIZE 16
; &core0 Trace.AutoFocus
; &core0 Trace.ShowFocus

;enable contextID for core #0 (LINUX)
; &core0 ETM.CONTEXTID 32
; &core0 TASK.Option THRCTX ON

;switch to MIX mode debugging
  &both MODE.MIX

ENDDO


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

4.4 amp_demo_start_core1.cmm

```
;dual-core debugging sample script for BORA (Zynq7000 based board)
```

```
;start_core1.cmm: start second instance of Trace32 (on demand)
;
;using INTERCOM port of current instance + 1 for communication
;

  &nodename=NODENAME()
  &intercomport_core0=FORMAT.DECIMAL(1.,intercom.port())
  &intercomport_core1=FORMAT.DECIMAL(1.,intercom.port()+1)

;Test if other software is already running or not
  if !intercom.ping(localhost:&intercomport_core1)
  (
    if "&nodename"==""
    (
      PRINT "assuming connection is USB"
      OS t32marm -c amp_config.t32 &intercomport_core1 TRACE32_B USB
CORE=2
    )
    else
    (
      PRINT "connection is ETH"
      OS t32marm -c amp_config.t32 &intercomport_core1 TRACE32_B NET
NODE=&nodename PACKLEN=1024 CORE=2
    )

    ;wait some time until the software finished booting
    ;if you get and intercom timeout error on the following framepos
commands, increase time
    WAIT 2.s
    INTERCOM.WAIT localhost:&intercomport_core1
  )
ENDDO
```