AXEL

Freescale i.MX6

Multicore ARM® Cortex®-A9

CPU Module

Axel Embedded Linux Kit (XELK)

AN-XELK-001

Asymmetric Multiprocessing (AMP) on Axel – Linux + FreeRTOS

<Page intentionally left blank>

Table of Contents

1 Preface

1.1 Abstract

Thanks to latest technological improvements, multicore processors are becoming popular in embedded world too. These architectures allows the implementation of processing schemes that were not feasible with traditional single-core CPUs. Among these, one of the most interesting is asymmetric multiprocessing (AMP for short). This configuration permits to address several requirements that the embedded system developers struggle to handle in case of single-core systems.

This application note describes in detail the implementation of Linux/FreeRTOS asymmetric multiprocessing configuration on **DAVE Embedded Systems** AXEL LITE. This configuration is a typical example about how to leverage AMP flexibility to combine, on one single piece of silicon, the versatility of Linux o.s. for general purpose computation, connectivity and HMI and the determinism of an RTOS to satisfy real-time constraints. Since AXEL family products are all based on Freescale i.MX6 processors, what here described applies to AXEL ULTRA too.

About debugging, AMP systems pose new challenges to software developers because multiple operating systems need to be debugged at the same time. The second part of this application note - written in collaboration with Lauterbach Srl - shows some advanced debugging techniques addressing these issues.

1.2 Copyrights/Trademarks

1.3 Standards

Dave SrL 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 AXEL CPU module, the AXEL EVB-Lite carrier board and the DACU carrier board.

AXEL 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-AXEL@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/dave-cpu-module-imx6-AXEL.html 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
i.MX6 Application Processor Reference Manual	http://cache.freescale.com/files/32bit/doc/ref_manual/IMX6DQRM.pdf?fpsp=1&WT_TYPE=Reference%20Manuals&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation
Freescale i.MX community webiste	https://community.freescale.com/community/imx
AXEL main page on **DAVE Embedded Systems** Developers Wiki	http://wiki.dave.eu/index.php/Category:AxelUltra
AXEL Hardware Manual	http://www.dave.eu/sites/default/files/files/axel-ultra-hm.pdf
XELK Quick Start Guide	http://wiki.dave.eu/index.php/XELK_Quick_Start_Guide
AXEL EVB-Lite page on **DAVE Embedded Systems** Developers Wiki	http://wiki.dave.eu/index.php/AxelEVB-Lite
DACU User's Guide	Provided with kit documentation

Tab. 1: Related documents

1.7 Conventions, Abbreviations, Acronyms

Abbreviation	**Definition**
AMP	Asymmetric multiprocessing
BTN	Button
EMAC	Ethernet Media Access Controller
GPI	General purpose input

Abbreviation	Definition
GPIO	General purpose input and output
GPO	General purpose output
LTIB	Linux Target Image Builder
PCB	Printed circuit board
PMIC	Power Management Integrated Circuit
PSU	Power supply unit
RTC	Real time clock
SOC	System-on-chip
SOM	System-on-module
WDT	Watchdog
XELK	AXEL Embedded Linux Kit

Tab. 2: Abbreviations and acronyms used in this manual

Revision History

Version	*Date*	*Notes*
0.9.0	June 2014	Draft
0.9.1	June 2014	Minor fixes
0.9.2	February 2015	Updated for XELK 2.0.0 release
0.9.3	April 2015	Minor fixes
0.9.4	January 2016	Minor fixes

2 Introduction

2.1 AXEL SOM family

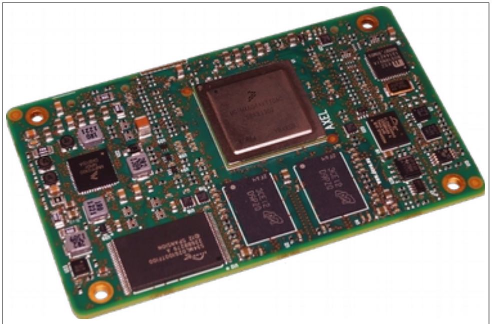


Fig. 1: AXEL ULTRA CPU module

AXEL is the new top-class Solo/Dual/Quad core ARM® Cortex®-A9 CPU module by **DAVE Embedded Systems**, based on the recent Freescale i.MX6 application processor.

Thanks to AXEL, customers have the chance to save time and resources by using a compact solution that permits to reach scalable performances that perfectly fits the application requirements avoiding complexities on the carrier board.

The use of this processor enables extensive system-level differentiation of new applications in many industry fields, where high-performance 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. AXEL offers great computational power, thanks to the rich set of peripherals, the Scalable ARM® Cortex®-A9 together with a large set of high-speed I/Os (up to 5GHz).

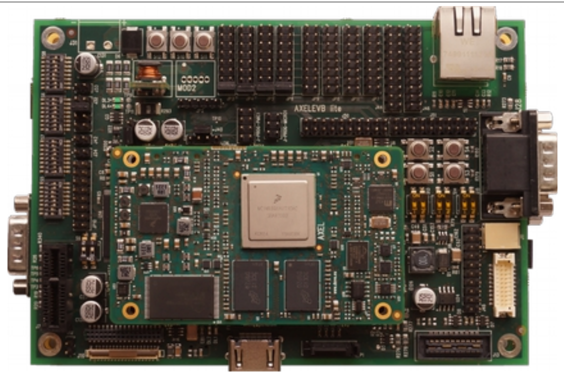


Fig. 2: AXEL plugged on AXEL EVB-LITE

AXEL enables designers to create smart products suitable for harsh mechanical and thermal environments, allowing the development of high computing and reliable solutions. Thanks to the tight integration

Fig. 3: AXEL LITE – Solo / Dual / Quad core ARM Cortex® A9

between the ARM Core-based processing system, designers are able to share the application through the multicore platform and/or to divide the task on different cores in order to match with specific application requirements (AMP makes possible the creation of applications where RTOS and Linux work together on different cores).Thanks to AXEL, customers are going to save time and resources by using a powerful and scalable compact solution, avoiding complexities on the carrier PCB.

AXEL family is composed by three models, one for each product line:

- AXEL ULTRA - http://www.dave.eu/products/AXEL
- AXEL ESATTA - http://www.dave.eu/products/AXEL-esatta
- AXEL LITE - http://www.dave.eu/products/AXEL-lite

For further information, please refer to AXEL Hardware Manual.

2.2 Freescale i.MX6 SOC

The i.MX6 Solo/Dual/Quad processors feature Freescale’s advanced implementation of the ARM® Cortex®-A9 MPCore, which operates at speeds up to 1.2 GHz. They include 2D and 3D graphics processors, 1080p video processing, and integrated power management. As a result, the i.MX6 devices are able to serve a wide range of applications including:

- Automotive driver assistance, driver information, and infotainment
- Multimedia-centric smart mobile devices
- Instrument clusters, and portable medical devices.
- E-Readers, smartbooks, tablets
- Intelligent industrial motor control, industrial networking, and machine vision
- IP and Smart camera
- Human-machine interfaces
- Medical diagnostics and imaging
- Digital signage
- Video and night vision equipment
- Multimedia-focused products
- Entertainment and gaming appliances

The i.MX6 application processor is composed of the following major functional blocks:

- ARM® Cortex®-A9 MPCore 2x/4x CPU Processor, featuring:
  - 1 Megabyte unified L2 cache shared by all CPU cores
  - NEON MPE coprocessor
  - General Interrupt Controller (GIC) with 128 interrupt support
  - Snoop Control Unit (SCU)

  - External memories interconnect
- Hardware accelerators, including:
  - VPU -Video Processing Unit
  - Two IPUv3H -Image Processing Unit (version 3H)
  - 2D/3D/Vector graphics accelerators
- Connectivity peripherals, including
  - PCIe
  - SATA
  - SD/SDIO/MMC
  - Serial buses: USB, UART, I^{2}C, SPI, ...

AXEL family can mount three versions of the i.MX6 processor. The following table shows a **comparison** between the processor models, highlighting the differences:

Processor	**# cores**	**Clock**	**L2 cache**	**DDR3**	**Graphics acceleration**	**IPU**	**VPU**	**SATA-II**
i.MX6 Solo	1	800 MHz 1 GHz	512 KB	32 bit @ 400 MHz	*3D*: Vivante GC880 *2D*: Vivante GC320 *Vector*: N.A.	1x	1x	N.A.
i.MX6 Dual	2	850 MHz 1 GHz 1.2 GHz	1 MB	64 bit @ 533 MHz	*3D*: Vivante GC2000 *2D*: Vivante GC320 *Vector*: Vivante GC335	2x	2x	Yes
i.MX6 Quad	4	850 MHz 1 GHz 1.2 GHz	1MB	64 bit @ 533 MHz	*3D*: Vivante GC2000 *2D*: Vivante GC320 *Vector*: Vivante GC335	2x	2x	Yes

Tab. 3: i.MX6 comparison

2.3 Asymmetric Multiprocessing

Thanks to latest technological improvements, multicore processors are becoming popular in embedded world too. These architectures allows the implementation of processing schemes that were not feasible with traditional single-core CPUs. Among these, one of the most interesting is asymmetric multiprocessing (AMP for short). This configuration permits to address several requirements that the embedded system developers struggle to handle in case of single-core systems.

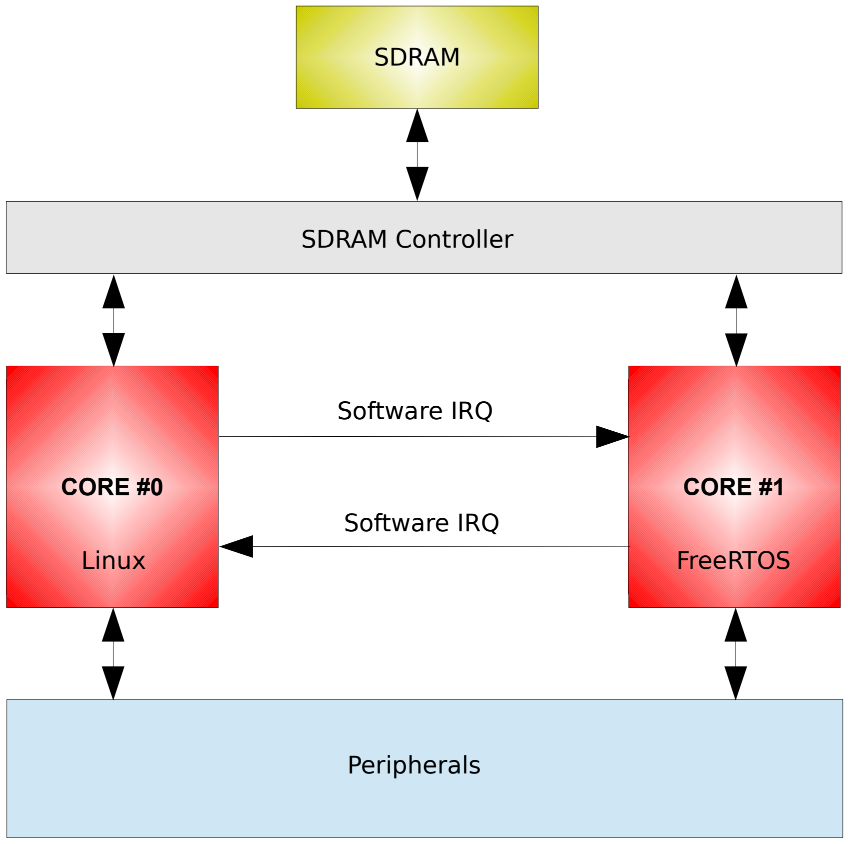
This application note describes in detail the implementation of Linux/FreeRTOS asymmetric multiprocessing configuration on **DAVE Embedded Systems** AXEL LITE SoM. This configuration is a typical example about how to leverage AMP flexibility to combine, on one single piece of silicon, the versatility of Linux o.s. for general purpose computation, connectivity and HMI and the determinism of an RTOS to satisfy real-time constraints. Since AXEL family products are all based on Freescale i.MX6 processors, what here described applies to AXEL ULTRA too.

As known, AMP allows a multicore system to run simultaneously[1] multiple Operating Systems (OS) that are independent of each other. In other words, each core has its own private memory space, which contains the OS and the applications that are to run on that core. In addition, there can be some shared memory space that is used for inter-core communication. This is in contrast to Symmetric Multiprocessing (SMP), in which one OS runs on multiple cores 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 to quickly implement applications that need to deliver deterministic, real-time responsiveness for markets such as automotive, industrial and others with similar requirements, while preserving the openness of Linux.

1 This is true simultaneity because multiple instructions are executed at the same time.

The following picture depicts the structure of the system.



Core #0:

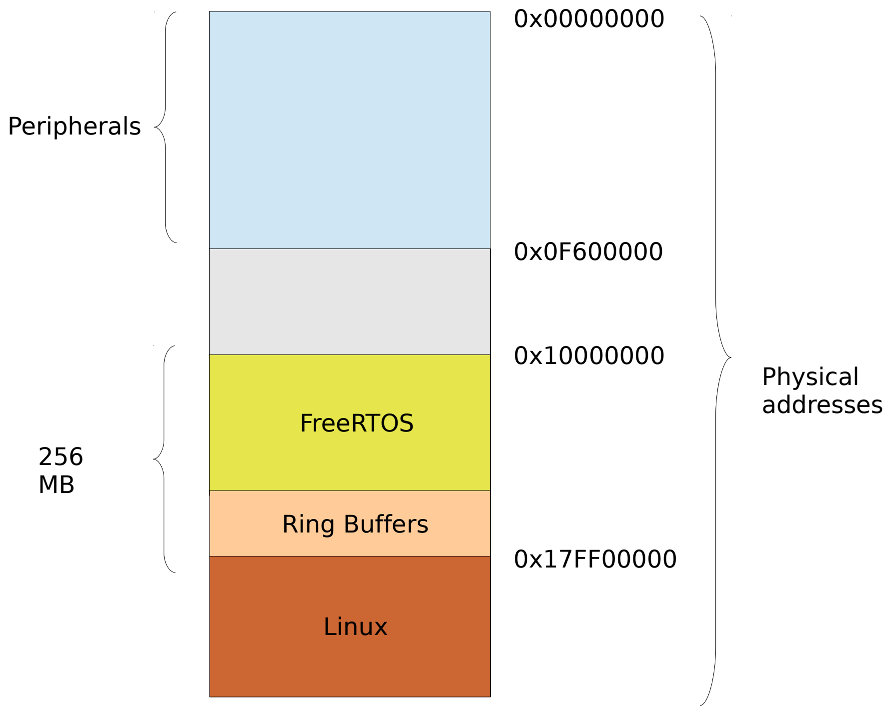
- takes care of boot process
- once Linux gets control of processor, initializes core #1 in SMP mode
- stops core #1 and switches it to AMP mode
- loads binary image of FreeRTOS that is then executed by core #1

2.3.1 Inter-core communication

Inter-core communication is based on RPMsg framework[2]. The adoption of a standardized and mainlined protocol improves dramatically the portability and the maintainability of the application code.

On Linux kernel, this framework is supported by the kernel 3.10.17_GA – released by Freescale itself along with L3.10.17_1.0.0_IMX6QDLS_BUNDLE BSP – upon which the XELK 2.0.0 is based[3].

The picture below shows how the system memory is fragmented. The portion of memory used to create a shared area between the two cores – ring buffers – is allocated inside the 256 MByte region used by FreeRTOS.



2 For more details please refer to http://omappedia.org/wiki/Category:RPMsg and https://www.kernel.org/doc/Documentation/rpmsg.txt.

3 For more details please see http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX6Q&fpsp=1&tab=Design_Tools_Tab.

2.4 XELK

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

- building the second stage bootloader (U-Boot)
- building and running Linux operating system on AXEL-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 i.MX6 development tools for building all the firmware/software components that will run on the target system.

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

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

3 AMP on AXEL

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.

The `latencystat` demo is a RPMsg-based application that exploits sophisticated techniques to handle inter-processors communication and synchronization.

3.1 Prerequisites

- AXEL Embedded Linux Kit (please refer to XELK Quick Start Guide for further details) version 2.0.0
- Access to AXEL git repositories (see below)
- An `arm-none-eabi-` toolchain for building FreeRTOS (please refer to Section 3.2.1)

3.1.1 Software components git repositories

The software components for this application note are provided as git repositories, so the user can immediately get access to the source trees and keep these components in sync and up to date with **DAVE Embedded Systems** repositories.

Component	Git Remote	Branch/Tag
Linux kernel	git@git.dave.eu:dave/axel/linux-2.6-imx.git	axel-feat-2.0.0-amp/AN-XELK-001-1.0.0
Freertos	git@git.dave.eu:dave/axel/freertos.git	axel/AN-XELK-001-1.0.0
Latency stat	git@git.dave.eu:dave/axel/latencystat.git	axel/AN-XELK-001-1.0.0

3.2 Building the software components

The following paragraphs describe how to build the software components required for this application. 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
- The GPIO7_13 pin (JP10.9 of the AXEL EVB-LITE) is toggled during the ISR
- The EIM_D19 signal (available on the R5 resistor on the AXEL EVB-LITE) is the output of the EPIT1 (which is the interrupt source)

The FreeRTOS application programs the EPIT1, which uses 66MHz as time-base (prescaler=1), setting FFFF.FFFF - 0x000F.4240 (1k tick) as the compare value. The timer starts a countdown from 0xFFFF.FFFF and, as it reaches the compare value, it triggers the ISR and toggles the EIM_D19 pin. The ISR counts the number of ticks after the compare value and saves that information to return it to the Linux application for the histogram calculation.

3.2.1 Build FreeRTOS

Building FreeRTOS is trivial. After cloning the repository, enter the following commands:

```
bash# cd freertos
bash# export PATH=<path to arm-eabi toolchain>/bin:
$PATH
bash# export ARCH=arm
bash# export CROSS_COMPILE=arm-none-eabi-
bash# make
```

The output of the build is contained into `output/mx6dq/Rpmsg-Demo/axellite_rev_a/Rpmsg-Demo.elf` (N.B: the `elf` file must be used, not the `bin`) e must be renamed as `freertos` and installed into the `/lib/firmware` directory of the root file system.

Please note that an `arm-none-eabi-` toolchain is required to build FreeRTOS. A suitable toolchain can be downloaded from https://launchpad.net/gcc-arm-embedded.

3.2.2 Build the latencystat application

The `latencystat` sources are available into the git repository. After cloning the repository, enter the following commands [4]:

```
bash# cd latencystat

bash# CC=arm-poky-linux-gnueabi-gcc make
```

The resulting binary must be copied from the building directory to the root file system.

3.2.3 Build Linux

To run this example, Linux kernel[5] must be rebuilt[6]. Thus configure the kernel using `imx_v7_axel_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 imx_v7_axel_amp_defconfig
make UIMAGE_LOADADDR=0x18008000 imx6q-xelk-l.dtb uImage
modules
[...]
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
….
….
  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, together with the file `arch/arm/boot/dts/imx6q-xelk-l.dtb`, which is the binary image of the device tree with the XELK hardware configuration.

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/mx6_remoteproc.ko
  LD [M]  drivers/rpmsg/rpmsg_freertos_statistic.ko
```

4 It is assumed that the development environment is already set up as described in XELK Quick Start Guide.
5 The kernel branch must be axel-feat-2.0.0-amp.
6 It is assumed that the development environment is already set up as described in XELK Quick Start Guide.

```
  LD [M]  drivers/rpmsg/virtio_rpmsg_bus.ko
  LD [M]  drivers/virtio/virtio.ko
  LD [M]  drivers/virtio/virtio_ring.ko
```

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

3.3 Running the demo applications

As stated before, this example shows a 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.

Once all the components are built, please boot the system and launch the following commands:

```
echo "now running with $(cat /proc/cpuinfo | grep
processor | wc -l) processors"


echo "start remote proc AMP"


insmod virtio.ko
insmod virtio_ring.ko
insmod virtio_rpmsg_bus.ko
insmod remoteproc.ko
insmod mx6_remoteproc.ko
insmod rpmsg_freertos_statistic.ko


echo "everything done"
echo "now Linux is running with $(cat /proc/cpuinfo |
grep processor | wc -l) processors"
```

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

```
root@axel:~# ./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.

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.

The following picture shows the AXEL EVB-LITE/DACU connected to Lauterbach PowerDebug Interface/USB3 via J18 connector.

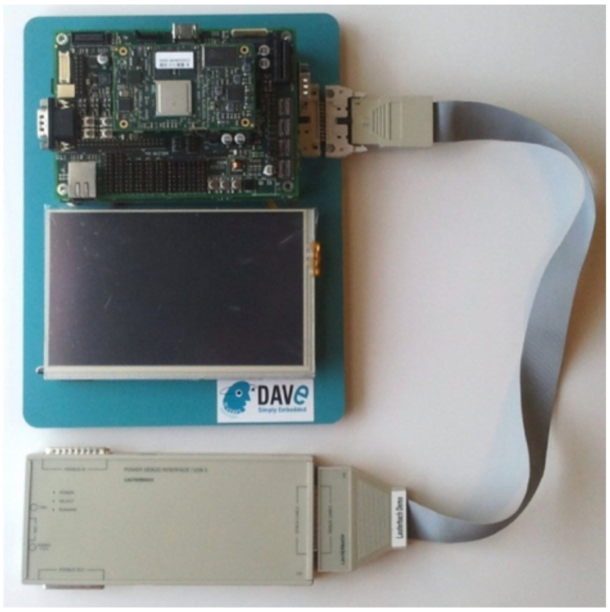


Fig. 4: AXELEVB-Lite/Dacu connected to Lauterbach PowerDebug Interface/USB3

The following sections describe in detail how to configure TRACE32 ® debugger to support debug of Linux running on the first i.MX6 core, and FreeRTOS, running on the second i.MX6 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:

```
config_mc.t32
```

The batch file starts automatically this startup script:

```
imx6-linux-rtos2.cmm
```

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

```
amp_start_core.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
&core2="intercom
localhost:&intercomport_core2"
&all_cores="GOSUB intercom_all "
```

where:

```
intercom_all:
  LOCAL &param
  ENTRY %Line &param
  &core0 &param
  &core2 &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
```

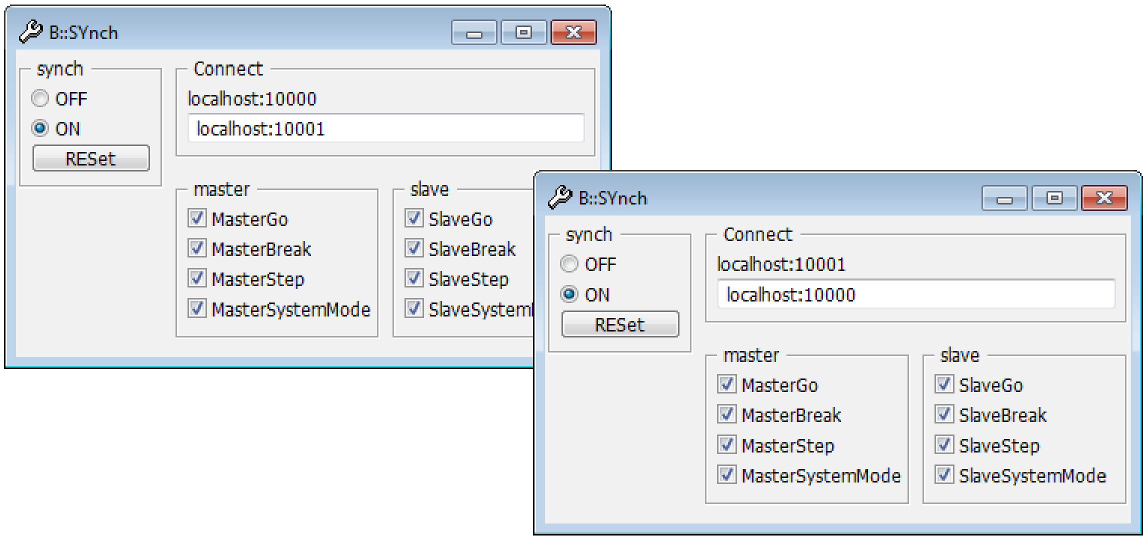
or:

```
&core0 SYStem.CONFIG.CORE 1 1

&core2 SYStem.CONFIG.CORE 3 1
```

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.

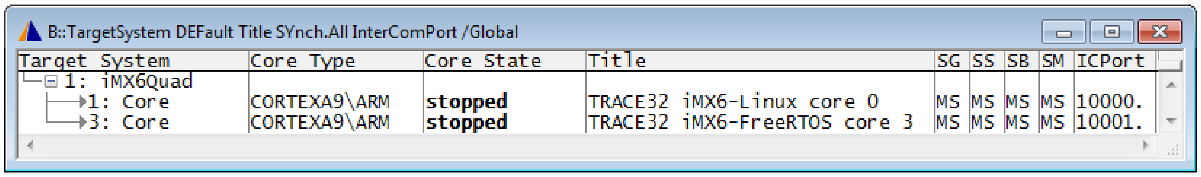
SlaveSystemMode ON

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.

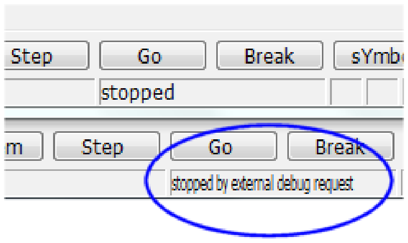
```
TargetSystem DEFault Title SYnch.All
InterComPort /Global
```


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

```
Target System       Core Type      Core State   Title                         SG SS SB SM ICPort
└─⊟ 1: iMX6Quad
    ├─▸1: Core      CORTEXA9\ARM   stopped      TRACE32 iMX6-Linux core 0     MS MS MS MS 10000.
    └─▸3: Core      CORTEXA9\ARM   stopped      TRACE32 iMX6-FreeRTOS core 3  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, i.e. the master debugger informs the slave debugger via socket communication on the host about specified events. On the other side, if the onchip 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 i.MX6 is automatically configured in TRACE32 SW.



stopped

stopped by external debug request

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.

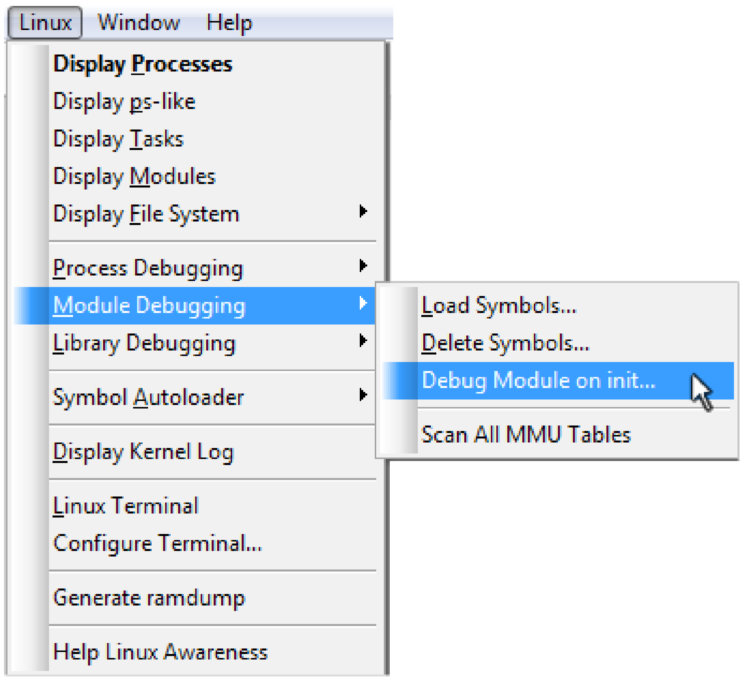
The Linux kernel awareness is configured with the commands:

```
; 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 0x80000000--
```

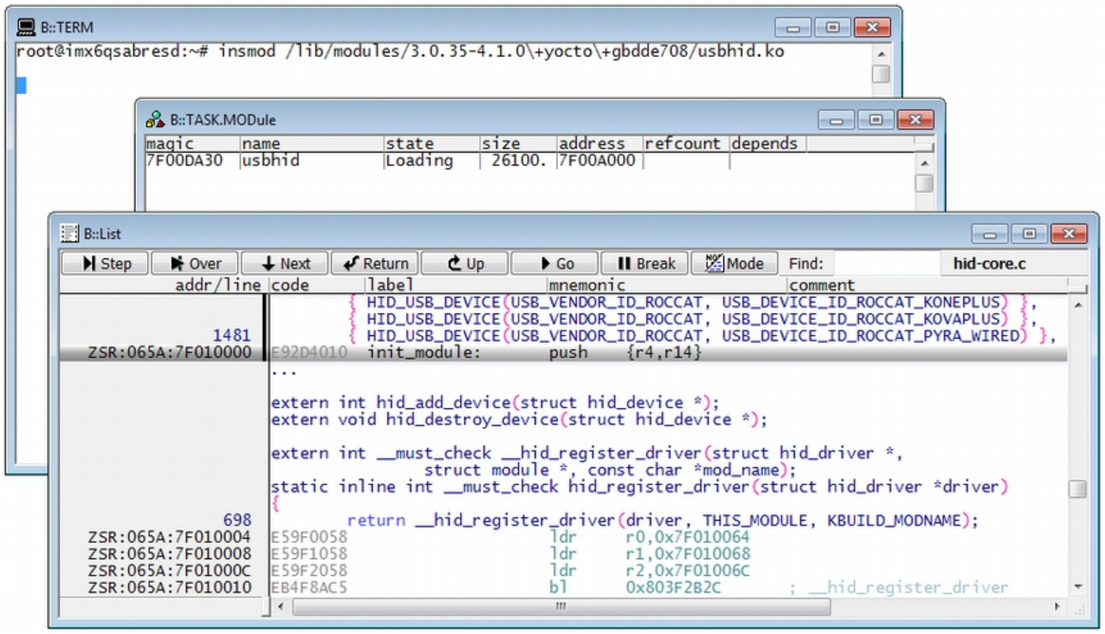
```
0x8fffffff 0x10000000
TRANSlation.Create 0x80000000--0x8fffffff
0x10000000
TRANSlation.COMMON 0x7f000000--0xffffffff
TRANSlation.TableWalk ON
TRANSlation.ON
```

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@imx6qsabresd:~# insmod /lib/modules/3.0.35-4.1.0\+yocto\+gbdde708/usbhid.ko
```

```
magic     name      state    size    address   refcount depends
7F00DA30  usbhid    Loading   26100. 7F00A000
```

```
          addr/line code      label            mnemonic          comment
                              { HID_USB_DEVICE(USB_VENDOR_ID_ROCCAT, USB_DEVICE_ID_ROCCAT_KONEPLUS) },
                              { HID_USB_DEVICE(USB_VENDOR_ID_ROCCAT, USB_DEVICE_ID_ROCCAT_KOVAPLUS) },
               1481           { HID_USB_DEVICE(USB_VENDOR_ID_ROCCAT, USB_DEVICE_ID_ROCCAT_PYRA_WIRED) },
  ZSR:065A:7F010000 E92D4010  init_module:     push    {r4,r14}
                    ...

                    extern int hid_add_device(struct hid_device *);
                    extern void hid_destroy_device(struct hid_device *);

                    extern int __must_check __hid_register_driver(struct hid_driver *,
                                    struct module *, const char *mod_name);
                    static inline int __must_check hid_register_driver(struct hid_driver *driver)
                    {
                698         return __hid_register_driver(driver, THIS_MODULE, KBUILD_MODNAME);
  ZSR:065A:7F010004 E59F0058                   ldr     r0,0x7F010064
  ZSR:065A:7F010008 E59F1058                   ldr     r1,0x7F010068
  ZSR:065A:7F01000C E59F2058                   ldr     r2,0x7F01006C
  ZSR:065A:7F010010 EB4F8AC5                   bl      0x803F2B2C        ; __hid_register_driver
```

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).

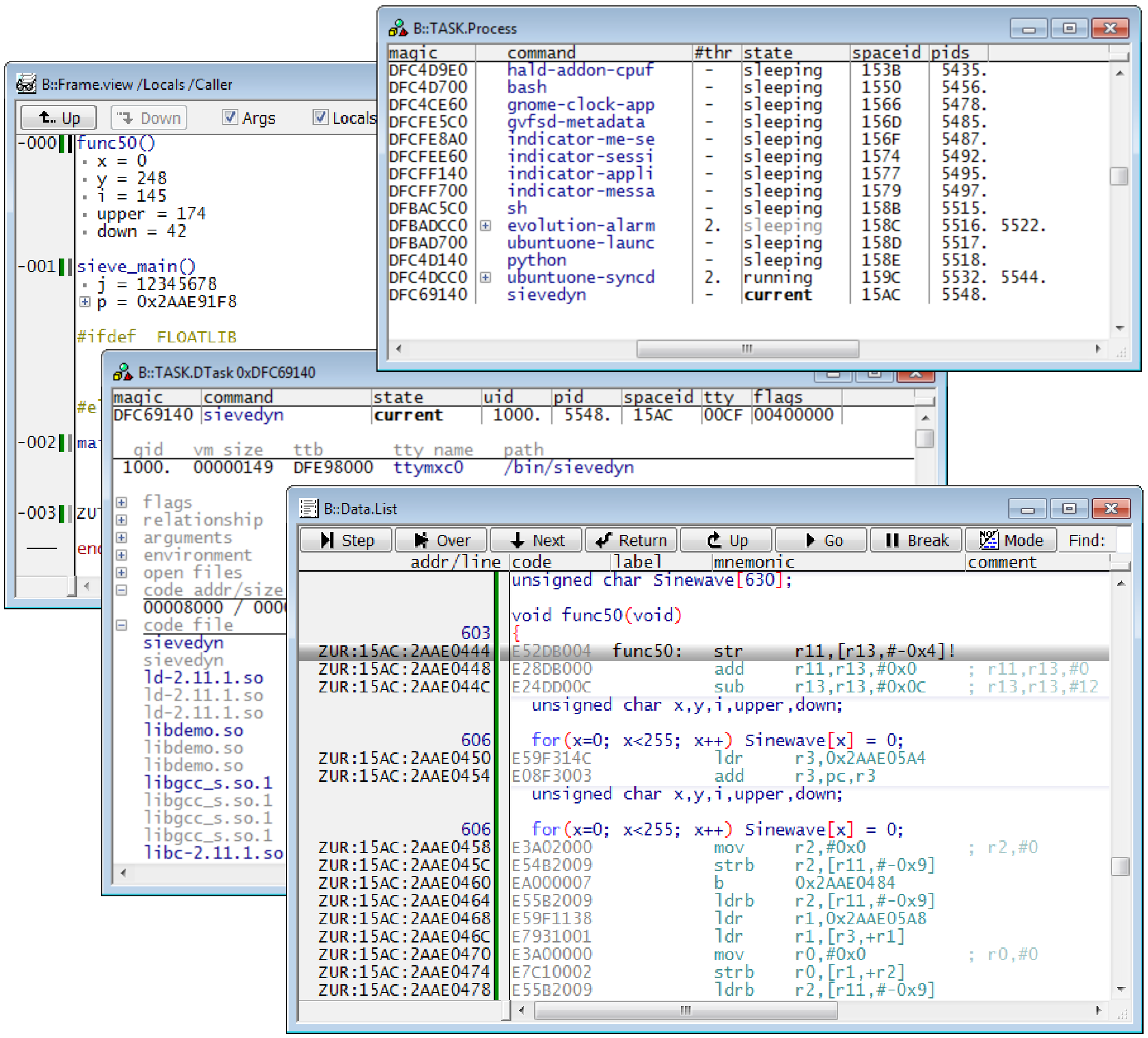
The Lauterbach debuggers also support debug of threads for multithreaded processes. In this case, the same address space is shared between different threads and the symbolic information can be loaded only once per process.

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
DFC4D9E0  hald-addon-cpuf   -   sleeping  153B    5435.
DFC4D700  bash              -   sleeping  1550    5456.
DFC4CE60  gnome-clock-app   -   sleeping  1566    5478.
DFCFE5C0  gvfsd-metadata    -   sleeping  156D    5485.
DFCFE8A0  indicator-me-se   -   sleeping  156F    5487.
DFCFEE60  indicator-sessi   -   sleeping  1574    5492.
DFCFF140  indicator-appli   -   sleeping  1577    5495.
DFCFF700  indicator-messa   -   sleeping  1579    5497.
DFBAC5C0  sh                -   sleeping  158B    5515.
DFBADCC0  evolution-alarm   2.  sleeping  158C    5516.  5522.
DFBAD700  ubuntuone-launc   -   sleeping  158D    5517.
DFC4D140  python            -   sleeping  158E    5518.
DFC4DCC0  ubuntuone-syncd   2.  running   159C    5532.  5544.
DFC69140  sievedyn          -   current   15AC    5548.
```

```
-000  func50()
        x = 0
        y = 248
        i = 145
        upper = 174
        down = 42

-001  sieve_main()
        j = 12345678
        p = 0x2AAE91F8

      #ifdef  FLOATLIB

      #e
-002  ma

-003  ZU
      en
```

```
magic     command    state    uid    pid    spaceid tty  flags
DFC69140  sievedyn   current  1000.  5548.  15AC    00CF 00400000

  gid    vm size   ttb       tty name  path
 1000.   00000149  DFE98000  ttymxc0   /bin/sievedyn

  flags
  relationship
  arguments
  environment
  open files
  code addr/size
   00008000 / 000
  code file
   sievedyn
   sievedyn
   ld-2.11.1.so
   ld-2.11.1.so
   ld-2.11.1.so
   libdemo.so
   libdemo.so
   libdemo.so
   libgcc_s.so.1
   libgcc_s.so.1
   libgcc_s.so.1
   libgcc_s.so.1
   libc-2.11.1.so
```

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

                     void func50(void)
                603  {
ZUR:15AC:2AAE0444    E52DB004  func50:  str    r11,[r13,#-0x4]!
ZUR:15AC:2AAE0448    E28DB000           add    r11,r13,#0x0    ; r11,r13,#0
ZUR:15AC:2AAE044C    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;
ZUR:15AC:2AAE0450    E59F314C           ldr    r3,0x2AAE05A4
ZUR:15AC:2AAE0454    E08F3003           add    r3,pc,r3
                       unsigned char x,y,i,upper,down;

                606    for(x=0; x<255; x++) Sinewave[x] = 0;
ZUR:15AC:2AAE0458    E3A02000           mov    r2,#0x0         ; r2,#0
ZUR:15AC:2AAE045C    E54B2009           strb   r2,[r11,#-0x9]
ZUR:15AC:2AAE0460    EA000007           b      0x2AAE0484
ZUR:15AC:2AAE0464    E55B2009           ldrb   r2,[r11,#-0x9]
ZUR:15AC:2AAE0468    E59F1138           ldr    r1,0x2AAE05A8
ZUR:15AC:2AAE046C    E7931001           ldr    r1,[r3,+r1]
ZUR:15AC:2AAE0470    E3A00000           mov    r0,#0x0         ; r0,#0
ZUR:15AC:2AAE0474    E7C10002           strb   r0,[r1,+r2]
ZUR:15AC:2AAE0478    E55B2009           ldrb   r2,[r11,#-0x9]
```

4.6 Setting up the FreeRTOS debug configuration

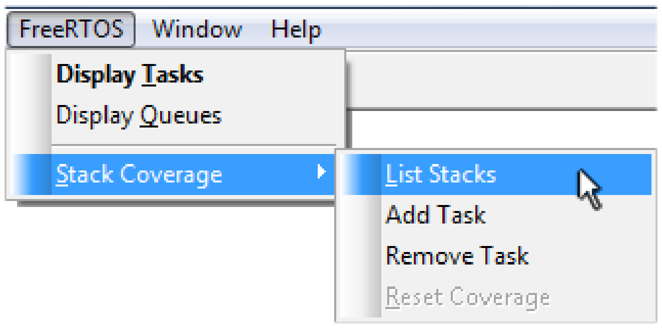
A similar kernel awareness concept as Linux, is provided by Lauterbach for many other RTOS. Among the others, also FreeRTOS is supported with the following awareness configuration.

```
; load FreeRTOS awareness
&core2 TASK.CONFIG
~~/demo/arm/kernel/freertos/freertos.t32


; load FreeRTOS menu
&core2 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 `&core2`. In this way, the second TRACE32 application will have visibility of FreeRTOS HLL debug info.

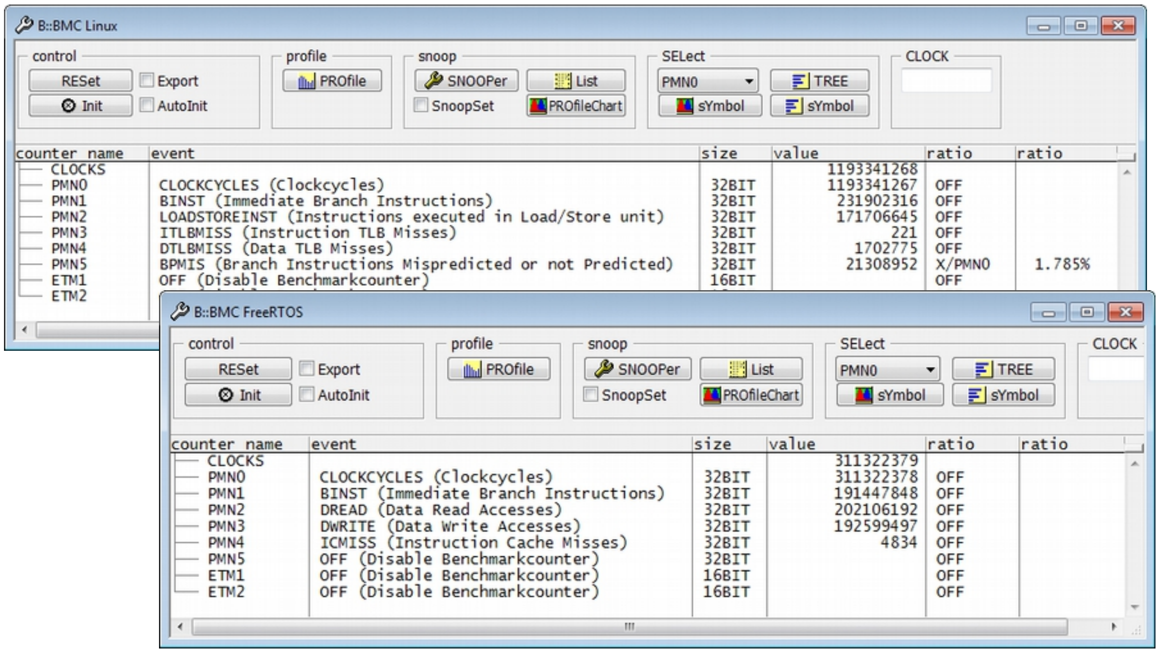
The FreeRTOS menu file (`freertos.men`) includes useful menu items developed for the TRACE32 GUI to ease FreeRTOS debugging.



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

4.7 Benchmark Counters

TRACE32 supports a full configuration and inspection of onchip benchmark counters implemented in i.MX6. All the counters can be configured in BMC window, and the analysis can be performed both during runtime and with stopped cores. Each i.MX6 core can be separately configured for any available counter. The counters can be cross-correlated each other, defining a ratio PMNx/PMNy in BMC window.



```
counter name  event                                                     size   value       ratio    ratio
── CLOCKS                                                                      1193341268
── PMN0       CLOCKCYCLES (Clockcycles)                                 32BIT  1193341267  OFF
── PMN1       BINST (Immediate Branch Instructions)                     32BIT   231902316  OFF
── PMN2       LOADSTOREINST (Instructions executed in Load/Store unit)  32BIT   171706645  OFF
── PMN3       ITLBMISS (Instruction TLB Misses)                         32BIT         221  OFF
── PMN4       DTLBMISS (Data TLB Misses)                                32BIT     1702775  OFF
── PMN5       BPMIS (Branch Instructions Mispredicted or not Predicted) 32BIT    21308952  X/PMN0     1.785%
── ETM1       OFF (Disable Benchmarkcounter)                            16BIT              OFF
── ETM2
```

```
counter name  event                                  size   value      ratio  ratio
── CLOCKS                                                   311322379
── PMN0       CLOCKCYCLES (Clockcycles)              32BIT  311322378  OFF
── PMN1       BINST (Immediate Branch Instructions)  32BIT  191447848  OFF
── PMN2       DREAD (Data Read Accesses)             32BIT  202106192  OFF
── PMN3       DWRITE (Data Write Accesses)           32BIT  192599497  OFF
── PMN4       ICMISS (Instruction Cache Misses)      32BIT       4834  OFF
── PMN5       OFF (Disable Benchmarkcounter)         32BIT             OFF
── ETM1       OFF (Disable Benchmarkcounter)         16BIT             OFF
── ETM2       OFF (Disable Benchmarkcounter)         16BIT             OFF
```

4.8 Onchip trace

The chip i.MX6 implements the ETB (Embedded Trace Buffer), a CoreSight hardware component providing onchip 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.

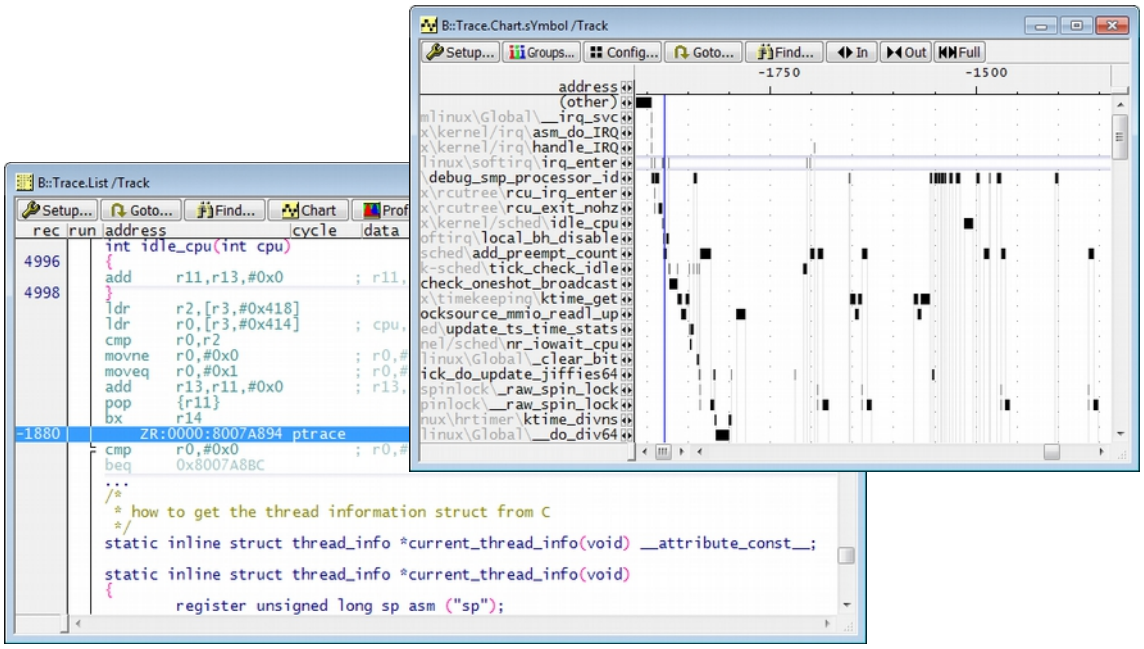
The trace, recorded for each core by each AMP TRACE32 GUI, can be synchronized using the XTRACK feature of TRACE32. In this way, an easy comparison is possible of program flow of each core at the same time.

The XTRACK feature is enabled with the following commands:

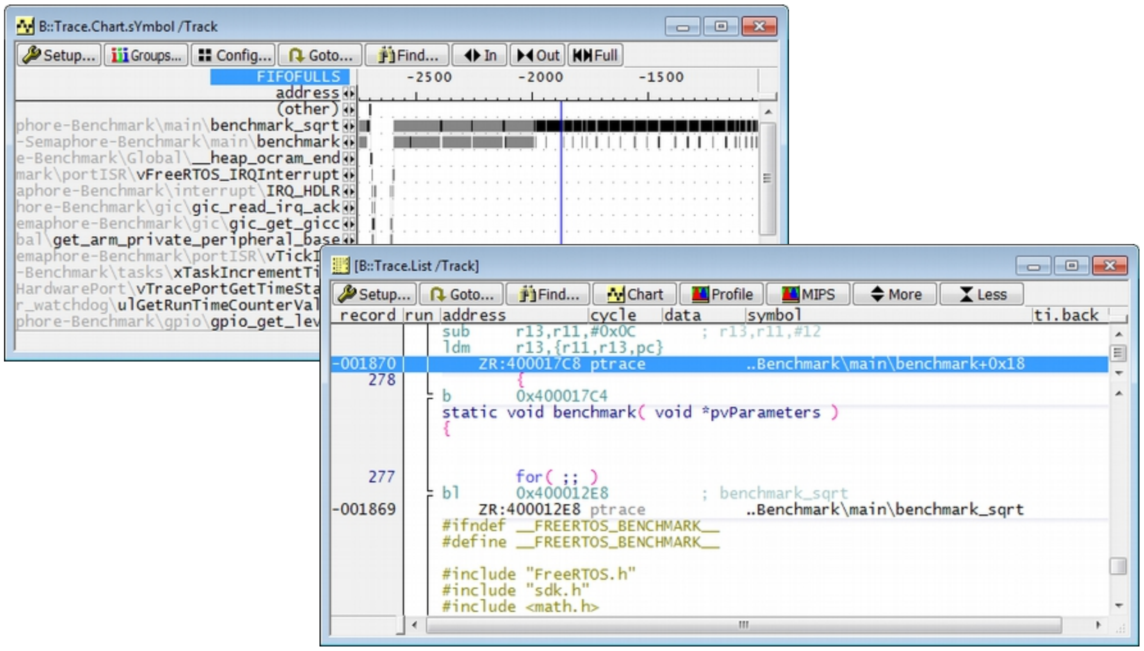
```
&core0 Synch.XTRACK
localhost:&intercomport_core2

&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:



```
 rec |run |address                          |cycle    |data
          int idle_cpu(int cpu)
 4996     {
          add      r11,r13,#0x0           ; r11,
 4998     }
          ldr      r2,[r3,#0x418]
          ldr      r0,[r3,#0x414]         ; cpu,
          cmp      r0,r2
          movne    r0,#0x0                ; r0,#
          moveq    r0,#0x1                ; r0,#
          add      r13,r11,#0x0           ; r13,
          pop      {r11}
          bx       r14
-1880          ZR:0000:8007A894 ptrace
          cmp      r0,#0x0                ; r0,#
          beq      0x8007A8BC
          ...
          /*
           * how to get the thread information struct from C
           */
          static inline struct thread_info *current_thread_info(void) __attribute_const__;

          static inline struct thread_info *current_thread_info(void)
          {
                   register unsigned long sp asm ("sp");
```

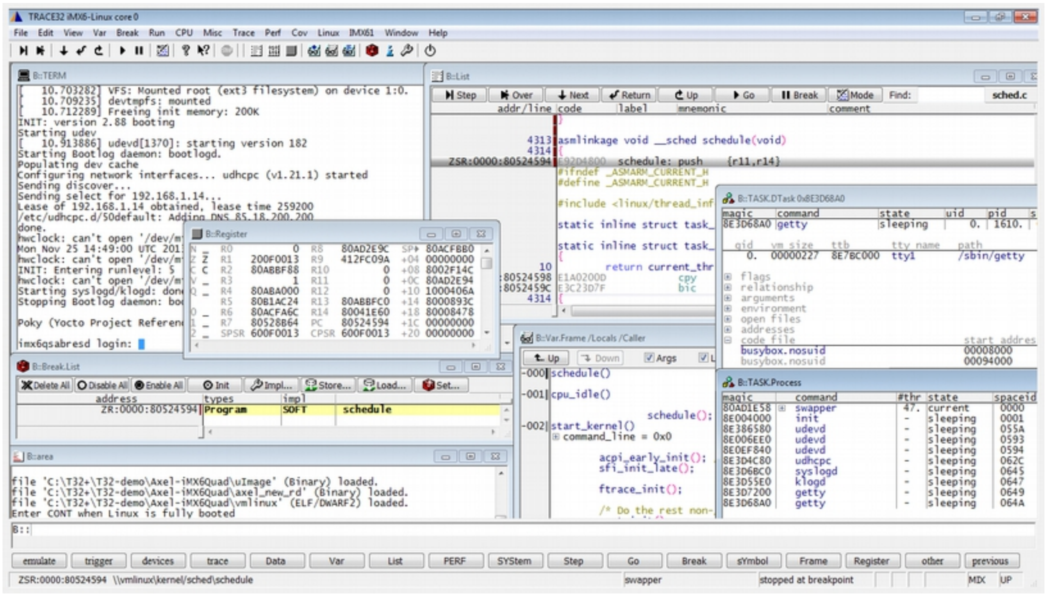

B::Trace.Chart.sYmbol /Track
Setup... Groups... Config... Goto... Find... In Out Full
FIFOFULLS
-2500
-2000
-1500
address
(other)
phore-Benchmark\main\benchmark_sqrt
-Semaphore-Benchmark\main\benchmark
e-Benchmark\Global\__heap_ocram_end
mark\portISR\vFreeRTOS_IRQInterrupt
aphore-Benchmark\interrupt\IRQ_HDLR
hore-Benchmark\gic\gic_read_irq_ack
emaphore-Benchmark\gic\gic_get_gicc
bal\get_arm_private_peripheral_base
emaphore-Benchmark\portISR\vTickI
-Benchmark\tasks\xTaskIncrementTi
HardwarePort\vTracePortGetTimeSta
r_watchdog\ulGetRunTimeCounterVal
phore-Benchmark\gpio\gpio_get_lev
[B::Trace.List /Track]
Setup... Goto... Find... Chart Profile MIPS More Less

```
record |run |address              |cycle   |data       |symbol                            |ti.back
             sub      r13,r11,#0x0C       ; r13,r11,#12
             ldm      r13,{r11,r13,pc}
-001870           ZR:400017C8 ptrace          ..Benchmark\main\benchmark+0x18
   278                {
             b        0x400017C4
             static void benchmark( void *pvParameters )
             {


   277                for( ;; )
             bl       0x400012E8           ; benchmark_sqrt
-001869           ZR:400012E8 ptrace          ..Benchmark\main\benchmark_sqrt
             #ifndef __FREERTOS_BENCHMARK__
             #define __FREERTOS_BENCHMARK__

             #include "FreeRTOS.h"
             #include "sdk.h"
             #include <math.h>
```

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

4.9 Summary view

In the pictures below, all the concepts previously discussed are shown as a summary global view of TRACE32 debugger, respectively for i.MX6 core 0 running a Linux kernel, and i.MX6 core 2 running a FreeRTOS based application.


TRACE32 iMX6-Linux core 0
File Edit View Var Break Run CPU Misc Trace Perf Cov Linux IMX61 Window Help
B::TERM
B::List
Step Over Next Return Up Go Break Mode Find: sched.c
B::Register
B::Break.List
Delete All Disable All Enable All Init Impl... Store... Load... Set...
B::area
B::Var.Frame /Locals /Caller
Up Down Args
B::TASK.DTask 0x8E3D68A0
B::TASK.Process
emulate trigger devices trace Data Var List PERF SYStem Step Go Break sYmbol Frame Register other previous

```
[   10.703282] VFS: Mounted root (ext3 filesystem) on device 1:0.
[   10.709235] devtmpfs: mounted
[   10.712289] Freeing init memory: 200K
INIT: version 2.88 booting
Starting udev
[   10.913886] udevd[1370]: starting version 182
Starting Bootlog daemon: bootlogd.
Populating dev cache
Configuring network interfaces... udhcpc (v1.21.1) started
Sending discover...
Sending select for 192.168.1.14...
Lease of 192.168.1.14 obtained, lease time 259200
/etc/udhcpc.d/50default: Adding DNS 85.18.200.200
done.
hwclock: can't open '/dev/m
Mon Nov 25 14:49:00 UTC 201
hwclock: can't open '/dev/m
INIT: Entering runlevel: 5
hwclock: can't open '/dev/m
Starting syslogd/klogd: don
Stopping Bootlog daemon: bo

Poky (Yocto Project Referen

imx6qsabresd login:
```

```
      addr/line code      label      mnemonic              comment
                 }
            4313 asmlinkage void __sched schedule(void)
            4314 {
ZSR:0000:80524594 E92D4800  schedule: push    {r11,r14}
                 #ifndef _ASMARM_CURRENT_H
                 #define _ASMARM_CURRENT_H

                 #include <linux/thread_inf

                 static inline struct task_

                 static inline struct task_
                 {
              10         return current_thr
:80524598 E1A0200D                cpy
:8052459C E3C23D7F                bic
            4314 {
```

```
N _  R0          0  R8    80AD2E9C  SP> 80ACFBB0
Z Z  R1   200F0013  R9    412FC09A  +04 00000000
C C  R2   80ABBF88  R10          0  +08 8002F14C
V _  R3          1  R11          0  +0C 80AD2E94
Q _  R4   80ABA000  R12          0  +10 1000406A
     R5   80B1AC24  R13   80ABBFC0  +14 8000893C
0 _  R6   80ACFA6C  R14   80041E60  +18 80008478
1 _  R7   80528B64  PC    80524594  +1C 00000000
2 _  SPSR 600F0013  CPSR  600F0013  +20 00000000
```

```
           address  types    impl
ZR:0000:80524594   Program  SOFT   schedule
```

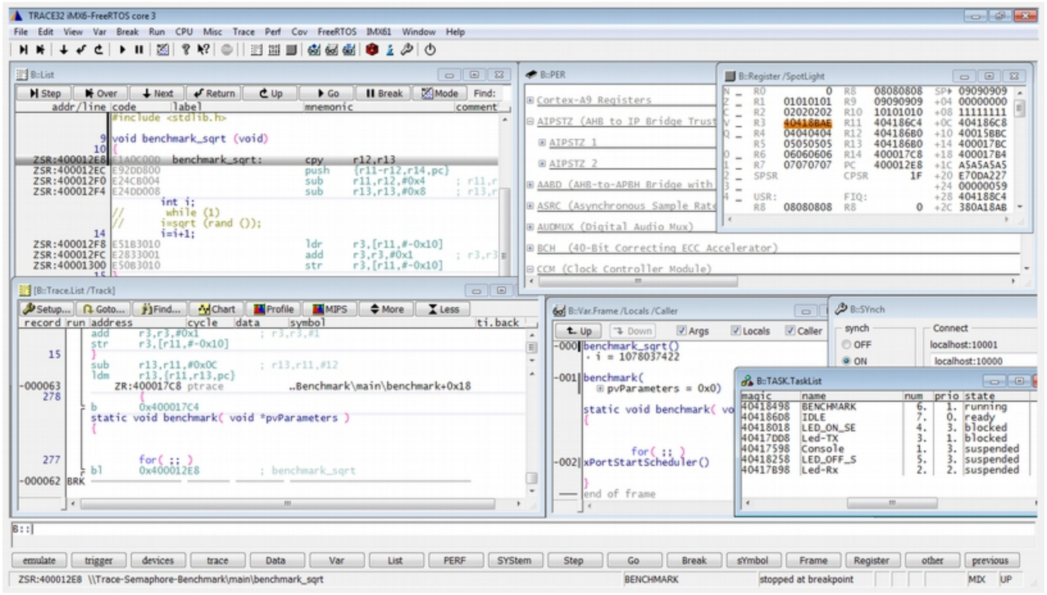
```
file 'C:\T32+\T32-demo\Axel-iMX6Quad\uImage' (Binary) loaded.
file 'C:\T32+\T32-demo\Axel-iMX6Quad\axel_new_rd' (Binary) loaded.
file 'C:\T32+\T32-demo\Axel-iMX6Quad\vmlinux' (ELF/DWARF2) loaded.
Enter CONT when Linux is fully booted
```

```
-000 schedule()

-001 cpu_idle()

                 schedule();

-002 start_kernel()
     ⊞ command_line = 0x0

          acpi_early_init();
          sfi_init_late();

          ftrace_init();

          /* Do the rest non-
```

```
magic     command  state     uid  pid
8E3D68A0  getty    sleeping   0.  1610.

  gid   vm size   ttb       tty name  path
   0.   00000227  8E7BC000  tty1      /sbin/getty

⊞ flags
⊞ relationship
⊞ arguments
⊞ environment
⊞ open files
⊞ addresses
⊟ code file                          start addres
  busybox.nosuid                     00008000
  busybox.nosuid                     00094000
```

```
magic      command   #thr state     spaceid
80AD1E58 ⊞ swapper    47. current   0000
8E004000   init        -  sleeping  0001
8E386580   udevd       -  sleeping  055A
8E006EE0   udevd       -  sleeping  0593
8E0EF840   udevd       -  sleeping  0594
8E3D4C80   udhcpc      -  sleeping  062C
8E3D6BC0   syslogd     -  sleeping  0645
8E3D55E0   klogd       -  sleeping  0647
8E3D7200   getty       -  sleeping  0649
8E3D68A0   getty       -  sleeping  064A
```

```
B::
ZSR:0000:80524594  \\vmlinux\kernel/sched\schedule    swapper    stopped at breakpoint    MIX  UP
```


TRACE32 iMX6-FreeRTOS core 3
File Edit View Var Break Run CPU Misc Trace Perf Cov FreeRTOS IMX61 Window Help
B::List
Step Over Next Return Up Go Break Mode Find:
B::PER
B::Register /SpotLight
[B::Trace.List /Track]
Setup... Goto... Find... Chart Profile MIPS More Less
B::Var.Frame /Locals /Caller
Up Down Args Locals Caller
B::SYnch
B::TASK.TaskList
emulate trigger devices trace Data Var List PERF SYStem Step Go Break sYmbol Frame Register other previous

```
      addr/line code      label            mnemonic              comment
                #include <stdlib.h>

              9 void benchmark_sqrt (void)
             10 {
ZSR:400012E8   E1A0C00D  benchmark_sqrt:  cpy    r12,r13
ZSR:400012EC   E92DD800                   push   {r11-r12,r14,pc}
ZSR:400012F0   E24CB004                   sub    r11,r12,#0x4     ; r11,r
ZSR:400012F4   E24DD008                   sub    r13,r13,#0x8     ; r13,r
                        int i;
                //       while (1)
                //      i=sqrt (rand ());
             14         i=i+1;
ZSR:400012F8   E51B3010                   ldr    r3,[r11,#-0x10]
ZSR:400012FC   E2833001                   add    r3,r3,#0x1       ; r3,r3
ZSR:40001300   E50B3010                   str    r3,[r11,#-0x10]
```

```
⊞ Cortex-A9 Registers
⊟ AIPSTZ (AHB to IP Bridge Trust
   ⊞ AIPSTZ 1
   ⊞ AIPSTZ 2
⊞ AABD (AHB-to-APBH Bridge with
⊞ ASRC (Asynchronous Sample Rate
⊞ AUDMUX (Digital Audio Mux)
⊞ BCH  (40-Bit Correcting ECC Accelerator)
⊟ CCM (Clock Controller Module)
```

```
N _  R0          0  R8    08080808  SP> 09090909
Z _  R1   01010101  R9    09090909  +04 00000000
C _  R2   02020202  R10   10101010  +08 11111111
V _  R3   40418BAE  R11   404186C4  +0C 404186C8
Q _  R4   04040404  R12   404186B0  +10 40015BBC
     R5   05050505  R13   404186B0  +14 400017BC
0 _  R6   06060606  R14   400017C8  +18 400017B4
1 _  R7   07070707  PC    400012E8  +1C A5A5A5A5
2 _  SPSR           CPSR        1F  +20 E70DA227
3 _                                 +24 00000059
4 _  USR:           FIQ:            +28 404188C4
     R8   08080808  R8           0  +2C 380A18AB
```

```
record run address        cycle   data    symbol                          ti.back
               add     r3,r3,#0x1          ; r3,r3,#1
               str     r3,[r11,#-0x10]
     15        }
               sub     r13,r11,#0x0C       ; r13,r11,#12
               ldm     r13,{r11,r13,pc}
-000063           ZR:400017C8 ptrace          ..Benchmark\main\benchmark+0x18
    278                {
               b       0x400017C4
               static void benchmark( void *pvParameters )
               {


    277                for( ;; )
               bl      0x400012E8          ; benchmark_sqrt
-000062 BRK
```

```
-000 benchmark_sqrt()
      · i = 1078037422

-001 benchmark(
      ⊞ pvParameters = 0x0)

     static void benchmark( vo
     {

             for( ;; )
-002 xPortStartScheduler()

     }
---- end of frame
```

```
synch        Connect
 ○ OFF       localhost:10001
 ◉ ON        localhost:10000
```

```
magic     name        num  prio state
40418498  BENCHMARK    6.   1.  running
404186D8  IDLE         7.   0.  ready
40418018  LED_ON_SE    4.   3.  blocked
40417DD8  Led-TX       3.   1.  blocked
40417598  Console      1.   3.  suspended
40418258  LED_OFF_S    5.   3.  suspended
40417B98  Led-Rx       2.   2.  suspended
```

```
B::
ZSR:400012E8  \\Trace-Semaphore-Benchmark\main\benchmark_sqrt    BENCHMARK    stopped at breakpoint    MIX  UP
```

4.10 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 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 config_mc.t32,imx6-linux-rtos2.cmm 10000 iMX6-Linux
NET NODE=%1 PACKLEN=1024 CORE=1
goto:eof

:useusb
start %t32path%\t32marm -c config_mc.t32,imx6-linux-rtos2.cmm 10000 iMX6-Linux
USB CORE=1
```

5.2 config_mc.t32

```
; Trace32 Configuration file for multicore debugging
;====================================================
;
; Parameters:
;  t32marm -c config_mc.t32[,startscript] <intercom_port> <title> <interface to
debug module> [interface options]
; Examples:
;  t32marm -c config_mc.t32,dualcoredemo.cmm 10000 Trace32_A NET NODE=pod-rei-s1
PACKLEN=1024 CORE=1
;  t32marm -c config_mc.t32,dualcoredemo.cmm 10000 CORE0 USB CORE=1

;Environment Variables
OS=
ID=T32${1}
TMP=C:\temp
SYS=C:\T32
HELP=C:\T32\pdf

;T32 API Access
; not used
```

```
;T32 Intercom
IC=NETASSIST
PORT=${1}

;Connection to Host
PBI=
${3}
${4}
${5}
${6}

;Screen Settings
SCREEN=
HEADER=Trace32 ${2}

;Printer Settings:
PRINTER=WINDOWS
```

5.3 imx6-linux-rtos2.cmm

```
; AMP dual-core debugging sample script for iMX6Quad based AXEL target
; AMP: one Trace32 instance per core
;
; Remark:
; Linux run on CORE #1
; FreeRTOS application run on CORE #3
;

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

RESet

; set up macros for intercom communication between the Trace32 instances
&nodename=NODENAME()
&intercomport_core0=FORMAT.DECIMAL(1.,intercom.port())
&intercomport_core2=FORMAT.DECIMAL(1.,intercom.port()+1)

; start Trace32 instance for second core
DO amp_start_core.cmm &intercomport_core2 3

&core0="" ;only to improve readability
&core2="intercom localhost:&intercomport_core2"
&all_cores="GOSUB intercom_all "

; arrange debugger windows
```

```
&core0 FramePOS 0.0 0.0 152. 38.
&core2 FramePOS 6.0 4.0 152. 38.
&all_cores WinPAGE.RESet
&all_cores WinCLEAR
&all_cores SCREEN.always

; set working path of core2 to match working path of core0
&core0 &path=OS.PWD()
&core2 cd "&path"

; reset settings
&all_cores SYnch.RESet
&all_cores SYStem.RESet
&all_cores sYmbol.RESet
&all_cores Break.RESet
&all_cores TRANSlation.RESet

; 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
&core0 MENU.RESet
&core0 do ~~/t32
&all_cores do toolbar_quit_all.cmm

; set titles
&core0 Title "TRACE32 iMX6-Linux core 0"
&core2 Title "TRACE32 iMX6-FreeRTOS core 3"

AREA.CLEAR
WinPOS 0.71429 29.0 80. 5. 0. 0. W001
AREA

SYStem.CPU iMX6Quad
CORE.ASSIGN 1
SYStem.CONFIG.CORE 1 1
SYStem.Option ResBreak OFF
SYStem.Option WaitReset 1.3s

SYStem.Option DACR ON          ; give Debugger global write permissions
TrOnchip.Set DABORT OFF        ; used by Linux for page miss!
TrOnchip.Set PABORT OFF        ; used by Linux for page miss!
TrOnchip.Set UNDEF OFF         ; may be used by Linux for FPU detection
SYStem.Up
```

```
Data.Set ASD:0x020d8000 %Long 0x00000521

; load linux kernel binary and Ramdisk
Data.LOAD.Binary uImage 0x12000000
Data.LOAD.Binary AXEL_rd 0x16000000

Go
TERM.Out 13. 13.
TERM.Out "setenv ramargs 'setenv bootargs ${bootargs} root=/dev/ram0 rw
ramdisk_size=16384'" 13.
WAIT 500.ms
TERM.Out "setenv mem 512M" 13.
WAIT 500.ms
TERM.Out "setenv addfreertos 'setenv bootargs ${bootargs} mem=${mem} maxcpus=1
loglevel=7 vmalloc=400M'" 13.
WAIT 500.ms
TERM.Out "setenv ipaddr 192.168.1.113" 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 "run addip" 13.
WAIT 500.ms
TERM.Out "run ramargs addcons addfreertos" 13.
WAIT 500.ms
Break

; run up to boot of uImage
Break.Set a:0x10008000 /Onchip
Go
TERM.Out "bootm ${loadaddr} 0x16000000" 13.
WAIT !STATE.RUN()
Break.Delete /ALL

; run up to boot of Linux
Break.Set a:0x10008000 /Onchip
Go
WAIT !STATE.RUN()
Break.Delete /ALL

SYStem.Option MMUSPACES ON     ; enable space ids to virtual addresses
Data.LOAD.Elf vmlinux /NoCODE /gnu /SPART 5.

; Declare the MMU format to the debugger
; - table format is "LINUX"
; - table base address is at label "swapper_pg_dir"
; - kernel address translation
; Map the virtual kernel symbols to physical addresses to give
; the debugger access to it before CPU MMU is initialized
MMU.FORMAT LINUX swapper_pg_dir 0x80000000--0x8fffffff 0x10000000 ;256Mb

; prepare debugger translation
TRANSlation.Create 0x80000000--0x8fffffff 0x10000000   ; map kernel pages at RAM
start
TRANSlation.COMMON 0x7f000000--0xffffffff              ; common area for kernel
and processes
TRANSlation.TableWalk ON       ; debugger uses a table walk to decode virtual
addresses
TRANSlation.ON                 ; switch on debugger(!) address translation

; Initialize Linux Awareness
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

Go
PRINT "Enter CONT when Linux is fully booted"
STOP
; issue a continue when linux is fully booted
Break

Data.Set a:0x020d8030 %long 0x400006d0
Data.Set a:0x020d8000 %long 0x00c00521

&all_cores SYStem.Down
&core0 CORE.ASSIGN 1
&core0 SYStem.CONFIG.CORE 1 1
&core0 SYStem.Mode attach

&core2 SYStem.Reset
&core2 SYStem.CPU iMX6Quad
&core2 SYStem.Option ResBreak OFF
&core2 SYStem.Option WaitReset 1.3s

&core2 SYStem.Option DACR ON          ; give Debugger global write permissions
&core2 TrOnchip.Set DABORT OFF        ; used by Linux for page miss!
&core2 TrOnchip.Set PABORT OFF        ; used by Linux for page miss!
&core2 TrOnchip.Set UNDEF OFF         ; may be used by Linux for FPU detection

&core2 CORE.ASSIGN 3
&core2 SYStem.CONFIG.CORE 3 1
&core2 SYStem.Mode attach
&core2 Break
```

```
&core2 Data.LOAD.Elf Trace-Semaphore-Benchmark.elf /RELOC .text AT 0x400006d0
/gnu /SPART 7.
&core2 sYmbol.SourcePATH.SetBaseDir "./freeRTOS-rc3"

; initialize RTOS support
&core2 print "initializing FreeRTOS support..."
&core2 TASK.CONFIG ~~/demo/arm/kernel/freertos/freertos           ; load
FreeRTOS awareness (freertos.t32)
&core2 MENU.ReProgram ~~/demo/arm/kernel/freertos/freertos        ; load
FreeRTOS menu (freertos.men)
&core2 HELP.FILTER.Add rtosfreertos   ; add FreeRTOS awareness manual to
filtered help

&core2 print "load complete."

&all_cores Onchip.RESet
&all_cores Trace.METHOD Onchip

; open some windows
&core2 WinPOS 0.0 0.0 70. 20. 16. 1.
&core2 Data.List

&all_cores WinPOS 89.0 2.0714 48. 9. 0. 0.
&all_cores Register.view /SpotLight
&all_cores WinPOS 89.125 15.929 48. 10. 0. 0. W003
&all_cores SYnch

; setup for synchronous step/go/break and system mode change
&all_cores SYNCH.RESet
&core0 SYNCH.CONNECT localhost:&intercomport_core2
&core2 SYNCH.CONNECT localhost:&intercomport_core0

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

&core0 TargetSystem DEFault Title SYnch.All InterComPort /Global

&core0 SYnch.XTRACK localhost:&intercomport_core2
&core2 SYnch.XTRACK localhost:&intercomport_core0

ENDDO
```

```
intercom_all:
  LOCAL &param
  ENTRY %Line &param
  &core0 &param
  ;&core1 &param
  &core2 &param
  RETURN

3.4. amp_start_core.cmm

; dual-core debugging reference script
;

ENTRY &intercomport_core &corenum

&patht32m=OS.PED()+"/t32marm"

;Test if other software is already running or not
 if !intercom.ping(localhost:&intercomport_core)
 (
    if "&nodename"==""
    (
      PRINT "assuming connection is USB"
      OS &patht32m -c config_mc.t32 &intercomport_core iMX6-FreeRTOS USB
CORE=&corenum
    )
    else
    (
      PRINT "connection is ETH"
      OS &patht32m -c config_mc.t32 &intercomport_core iMX6-FreeRTOS NET
NODE=&nodename PACKLEN=1024 CORE=&corenum
    )

    ;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_core
 )

ENDDO
```

5.4 U-Boot environment

The following is the u-boot environment that can be printed using the `print` command (please note this is an example and that the actual variables may differ for different U-Boot settings):

```
addcons=setenv bootargs ${bootargs} console=${console},115200n8 mem=${kernel_memory} cma=16M debug
maxcpus=${nr_cpus}
addip=setenv bootargs ${bootargs} ip=${ipaddr}:${serverip}:${gateway}:${netmask}:${hostname}:$
{ethdev}
baudrate=115200
bootcmd=run net_nfs
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/hello_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_memory=1008M
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-1.0.1
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.ima
ge.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_h=sf probe 0 0 0;sf erase ${header_base} 0x40000;sf write ${loadaddr} ${header_base} 0x40000

Environment size: 2721/131067 bytes
```