1. Who are Lauterbach
2. TRACE32® Overview
3. TRACE32® Kernel Awareness
4. Usage Examples
1) Who are Lauterbach

- Largest Manufacturer of debuggers worldwide
- Founded in 1979
- Based in Höhenkirchen, near Munich
- Privately owned by the founders
- Approx. 120 employees worldwide, with subsidiaries in China, France, Italy, Japan, Tunisia, UK, USA
- Other territories covered by exclusive highly technical distributors
The Lauterbach Difference

- Company is privately owned and engineering led
  - No chasing quarterly results or kow-towing to shareholders
  - >80% worldwide staff are engineers
- All R&D, Engineering, and Production takes place at our facility outside Munich
- Excellent reputation for providing timely, high quality support
  - Even Mr. Lauterbach still answers support calls!
- We only make debuggers
  - We have to work with all compilers, RTOS, 3rd party tools, etc.
  - No dilution of effort
- Long-term close relationships with silicon vendors
  - Support for tens of thousands of devices from approx. 75 silicon vendors!

2) TRACE32® Tool Overview
Modular tools designed to Grow

Debug tools

+ 

= 

Debug and Trace tools

+ 

= 

TRACE32 and RIOT OS
Modular tools designed to Grow

Digital and Analogue Logic Trace tools

3) TRACE32® Kernel Awareness
What is a Kernel Awareness

> Extension to the TRACE32® debugger
  > Currently over 80 RTOS' supported
  > All delivered free of charge (included on DVD or in software download image)
> Loaded at runtime
  > Two files: kernel awareness and menu to access features
  > Some optional scripts to simplify complex operations
> Provides access to RTOS resources at runtime
  > Display system objects, such as tasks, threads, semaphores, mailboxes, etc.
  > Set task aware breakpoints
  > Task aware performance monitoring
  > Task aware tracing
> May be built by Lauterbach, a TRACE32 user, or the RTOS developer

To create a kernel awareness plugin

> Requires the Extension Development Kit (EDK)
  > Free of Charge
  > Signed NDA required
  > Supports Windows and Linux build hosts
  > I used Fedora Core 31
> EDK contains
  > C Library Routines
  > Make files
  > Custom Embedded C Cross compiler
  > Documentation
  > Examples
Build Process Overview

- Take existing example and adapt it
  - Much easier than starting from scratch
  - Makefile and build environment already set
  - Skeleton functions exist for most OS objects
- A few mandatory functions need to be provided
  - Info about current task/thread
  - List of all tasks/threads
  - Details of registers saved/restored during a context switch
- Everything else is optional
  - All of the optional components are defined in the main awareness file
  - Define new commands
  - Define new functions
  - Define anything else to make the user’s life easier when debugging your kernel/RTOS

Other Requirements

- Header files and some source files for RTOS
  - Documentation and debug compiled kernel may be used
- RIOT OS is provided in source
  - Header files and Source files are well documented
  - Very helpful and knowledgeable community
- Working build environment
  - To create example applications to test the awareness against
  - Most of this can often be performed in a simulated environment, using TRACE32®
- Supported hardware target
  - Final testing on real hardware with real tools 😊
4) Usage Examples

New Menu

- The menu file is part of the awareness
  - Added to the UI after the awareness has been loaded
  - Provides convenient access to many OS specific views
Task and Thread Lists

- Display a list of active tasks and threads
- Where target supports dual port memory, lists are dynamic
- ‘magic’ column has a right-click menu giving access to extra information about each task/thread

Access to System Objects
Switch between tasks/threads

- Handy dropdown on status bar to quickly switch between task or threads
- All open windows (unless otherwise anchored) will switch to the new context.
  - Source listing
  - Registers
  - Variables

Stack frame for each Task/thread

- View stack usage for each task/thread
  - Supports standard and non-standard stack pre-fill values
Stack frame for each Task/thread

- View call stack for each task or thread
- Walk up and down the call stack – all relevant open windows change their view(s)

Task/Thread Aware Breakpoints

- Use dropdown to set task or thread aware breakpoints
JTAG based task/thread profiling

- Use whatever features the CPU provides
- If none, use Stop&Go
- May be some level of intrusion
  - Runtime will be indicated on display

Trace based task/thread profiling

- Highly accurate task/thread runtime profiling
- Timeline view
Trace based task/thread profiling

- Raw task/thread switch data
- Thread magic
- Time since last switch
- Time to next switch

THANK YOU!

Richard Copeman
richard.copeman@lauterbach.com