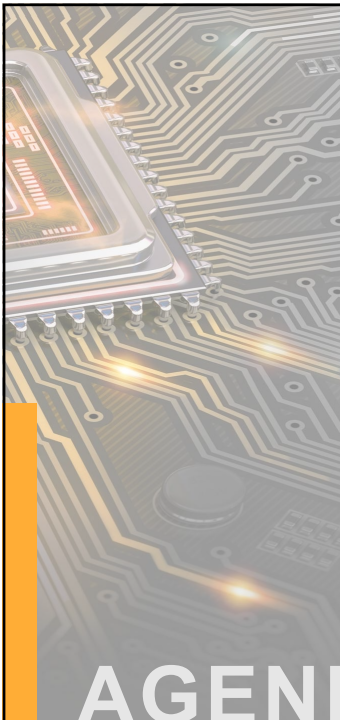




Debug and Profile with TRACE32® and RIOT OS

Richard Copeman | richard.copeman@lauterbach.com

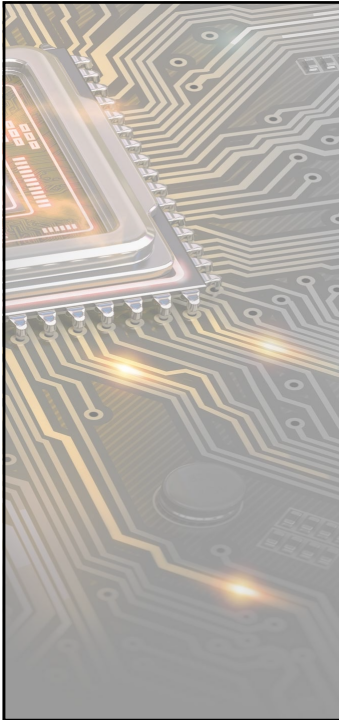
1



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

AGENDA

2



1) Who are Lauterbach

3

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



4

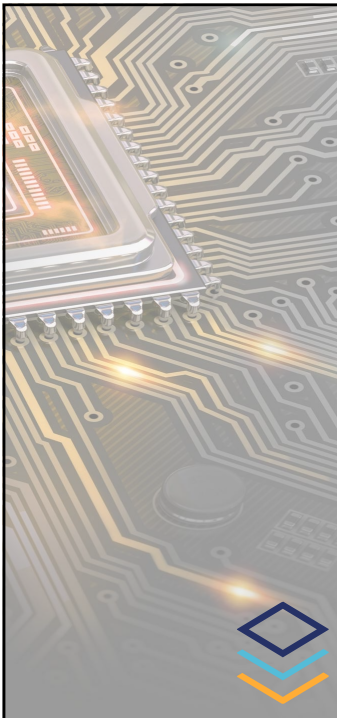
4

The Lauterbach Difference

- Company is privately owned and engineering led
 - No chasing quarterly results or kow-towing to share holders
 - >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!

5

5

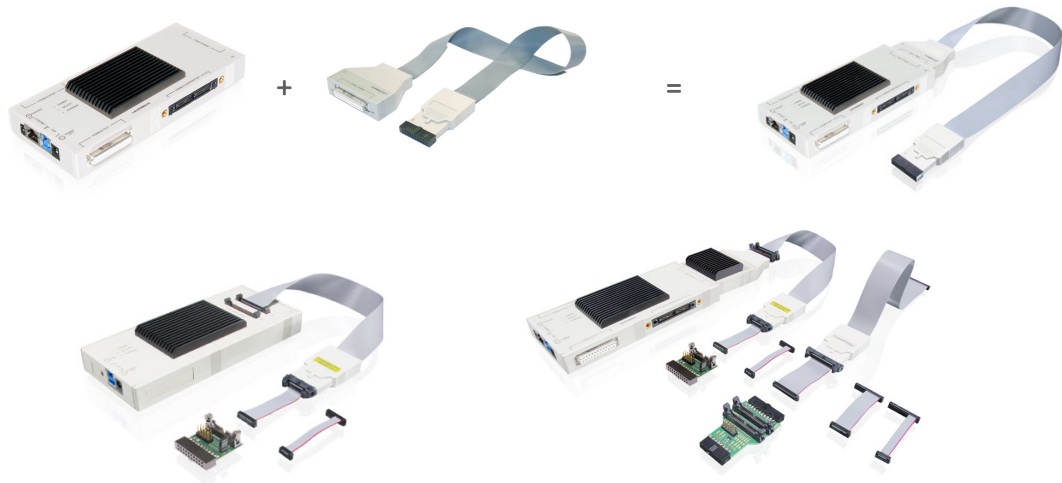


2) TRACE32® Tool Overview

6

Modular tools designed to Grow

Debug tools

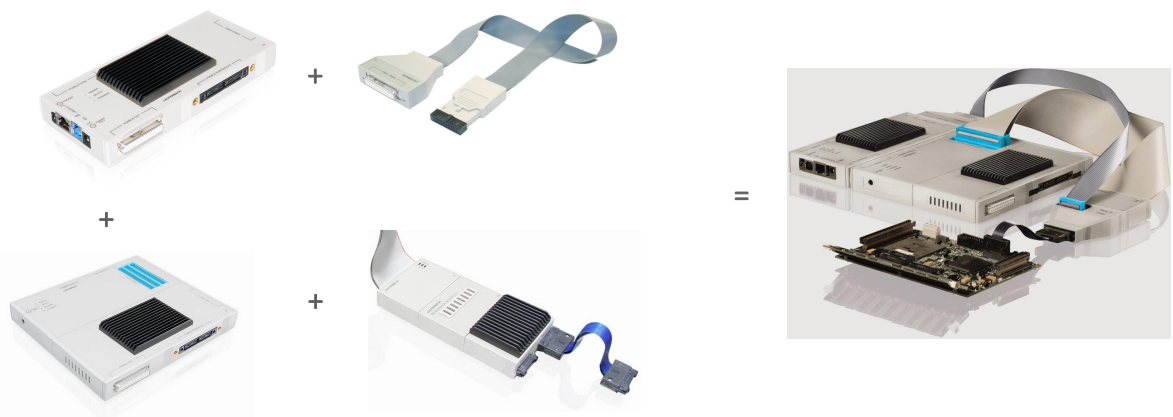


7

7

Modular tools designed to Grow

Debug and Trace tools

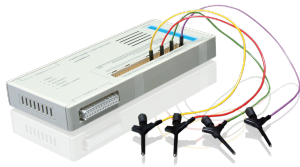


8

8

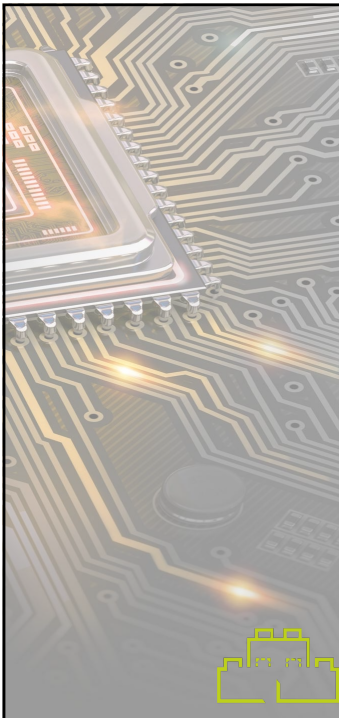
Modular tools designed to Grow

Digital and Analogue Logic Trace tools



9

9



3) TRACE32® Kernel Awareness

10

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

11

11

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

12

12

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

13

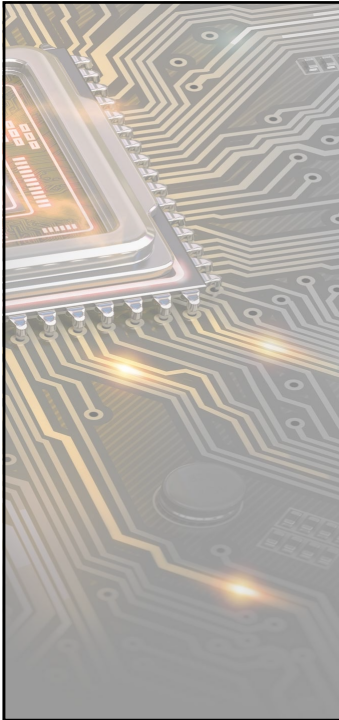
13

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 😊

14

14

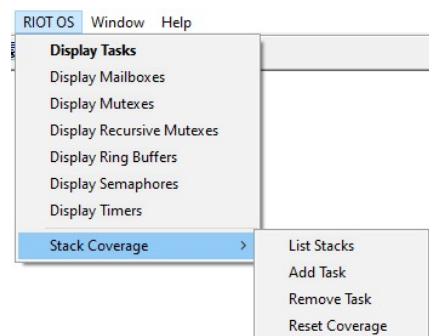


4) Usage Examples

15

New Menu

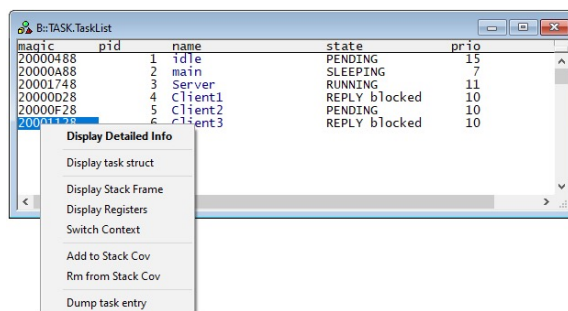
- The menu file is part of the awareness
 - Added to the UI after the awareness has been loaded
 - Provides convenient access to many OS specific views



16

Task and Thread Lists

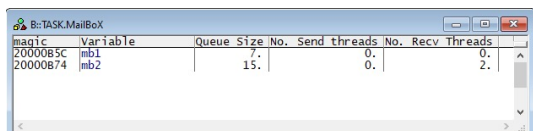
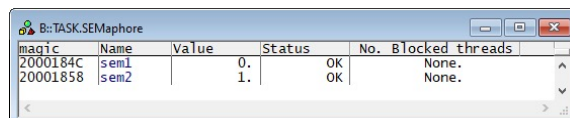
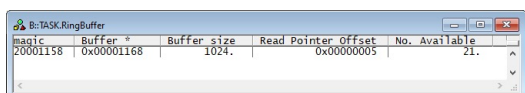
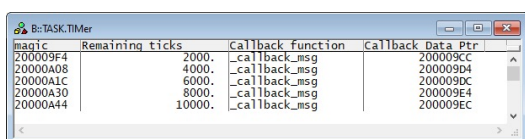
- > Display a list of active tasks and threads
 - > Where target supports dual port memory, lists are dynamic
 - > 'magic' column has a right-click menu giving access to extra information about each task/thread



17

17

Access to System Objects

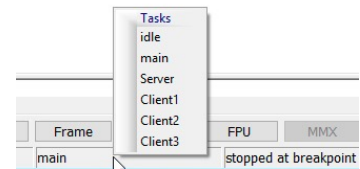


18

18

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



19

19

Stack frame for each Task/thread

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

The image shows a debugger window titled 'B:TASK.Stack.view' displaying a table of stack usage for various tasks. The table has columns for 'name', 'low', 'high', 'sp', '% lowest', 'spare', and 'max'. The data is as follows:

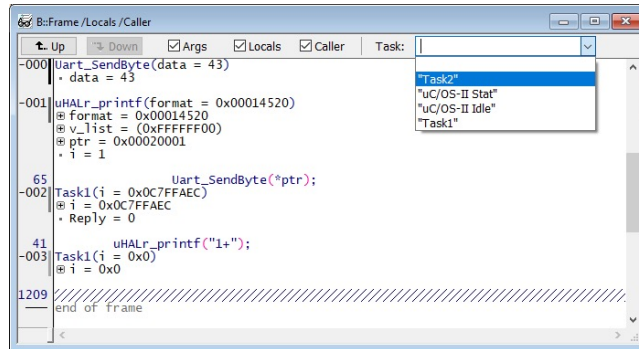
name	low	high	sp	% lowest	spare	max
idle	200003B8	200004B8	20000470	28%	20000470	000000B8 28%
main	200004B8	20000AB8	200009EC	13%	200009EC	00000534 13%
pong	2000124C	2000184C	200017AC	10%	200017AC	00000560 10%
Mtx_test	2000104C	2000124C	200011A4	32%	200011A4	00000158 32%
Mtx_test_2	20000C4C	20000E4C	20000DA4	32%	20000DA4	00000158 32%
Mtx_test_3	20000E4C	2000104C	20000F84	39%	20000F84	00000138 39%

20

20

Stack frame for each Task/thread

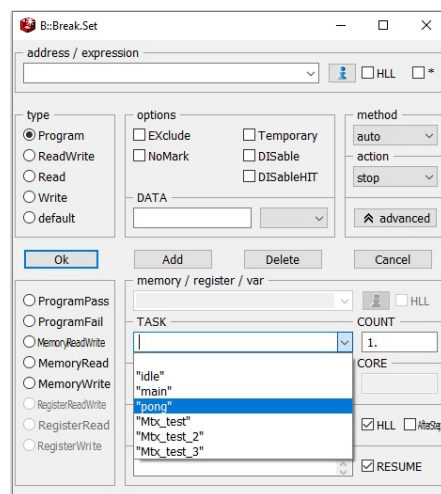
- > View call stack for each task or thread
 - > Walk up and down the call stack – all relevant open windows change their view(s)



21

Task/Thread Aware Breakpoints

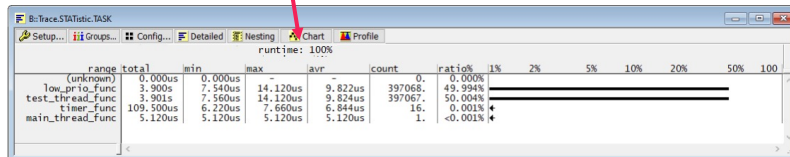
- > Use dropdown to set task or thread aware breakpoints



22

JTAG based task/thread profiling

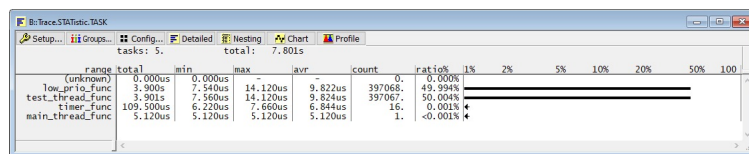
- > Use whatever features the CPU provides
 - > If none, use Stop&Go
 - > May be some level of intrusion
 - > Runtime will be indicated on display



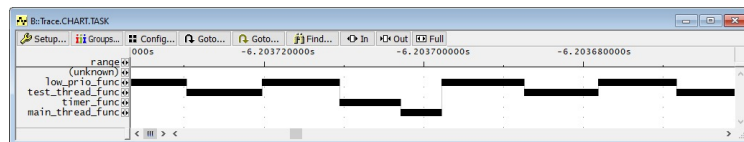
23

Trace based task/thread profiling

- > Highly accurate task/thread runtime profiling



- > Timeline view



24

Trace based task/thread profiling

➤ Raw task/thread switch data

The main window shows a trace of task/thread switches with columns: record, address, cycle, data, t.t.back, and t.t.fore. Red arrows point from the text labels to the corresponding columns in the trace table.

record	address	cycle	data	t.t.back	t.t.fore
-001866817	D:20000714	wr-long	20000A30	9.320us	9.780us
-0018668391	D:20000714	wr-long	20000800	9.780us	10.340us
-0018668364	D:20000714	wr-long	20000A30	10.340us	9.340us
-0018668337	D:20000714	wr-long	20000800	9.340us	9.780us
-0018668311	D:20000714	wr-long	20000A30	9.780us	10.320us
-0018668284	D:20000714	wr-long	20000800	10.320us	9.340us
-0018668257	D:20000714	wr-long	20000A30	9.340us	10.340us
-0018668224	D:20000714	wr-long	20000800	10.340us	9.320us
-0018668197	D:20000714	wr-long	20000A30	9.320us	10.220us
-0018668170	D:20000714	wr-long	20000800	10.220us	9.900us
-0018668144	D:20000714	wr-long	20000A30	9.900us	9.320us
-0018668117	D:20000714	wr-long	20000800	9.320us	10.340us
-0018668090	D:20000714	wr-long	20000A30	10.340us	9.780us
-0018668064	D:20000714	wr-long	20000800	9.780us	9.340us
-0018668037	D:20000714	wr-long	20000A30	9.340us	10.320us
-0018668010	D:20000714	wr-long	20000800	10.320us	9.780us
-0018667982	D:20000714	wr-long	20000A30	9.780us	9.780us
-0018667950	D:20000714	wr-long	20000800	9.780us	9.440us
-0018667923	D:20000714	wr-long	20000A30	9.440us	10.220us
-0018667896	D:20000714	wr-long	20000800	10.220us	9.900us
-0018667870	D:20000714	wr-long	20000A30	9.900us	9.320us
-0018667843	D:20000714	wr-long	20000800	9.320us	10.240us
-0018667816	D:20000714	wr-long	20000A30	10.240us	9.880us
-0018667790	D:20000714	wr-long	20000800	9.880us	9.340us
-0018667761	D:20000714	wr-long	20000A30	9.340us	10.220us

The task list window shows the following data:

magic	name	state	prio
20000800	test_thread_func	suspended	17.
2000088C	sem_func1	ready	21.
20000718	IdleThread	ready	255.
20000A30	low_prio_func	running	17.
20000918	sem_func2	semaphore	17.
200004BC	timer_func	semaphore	15.
20000A30	low_prio_func	mutex	17.
200009A4	q_thread_func	queue RCV	15.

25



THANK YOU!

Richard Copeman
richard.copeman@lauterbach.com

26