

# TRACE32<sup>®</sup>



## Debug & Trace

ARM<sup>®</sup>



## ARM® support at a glance

More than 18 years of experience in ARM debugging enable us to provide the best-in-class debug and trace tools for ARM based systems:

- Multicore debugging and tracing for any mixture of ARM and DSP cores
- Support for all CoreSight components to debug and trace an entire SoC
- Powerful code coverage and run-time analysis of functions and tasks
- OS-aware debugging of kernel, libraries, tasks of all commonly used OSes

## Debugging ARM®-based Systems

The Lauterbach debug tools for ARM accompany you throughout the whole development process, from the early pre-silicon phase by debugging on an instruction set simulator or a virtual prototype, over board bring-up, till quality and maintenance work on the final product.

The features range from simple step/go/break, programming of on-chip flash, external NAND, eMMC, parallel and serial NOR flash devices, support for NEON and VFP units, to OS-aware debug and trace concepts for 32-bit and 64-bit multicore systems.

**Peripheral register**

PMC (Power Management Controller)							
LVDS1	10	LVDF	Not detected	LVDACK	NACK	LVDIE	Disabled
		LVDRE	Enabled	LVDV	Low trip		
LVDS2	20	LVWF	Not detected	LWACK	NACK	LWIE	Enabled
		LVWV	Low trip point				
REGSC	04	VLPRS	Not VLPR				
		REGONS	Run				

SD:4007D001 0--1 Low-Voltage Warning

The peripheral registers on the chip are displayed on a logical level. The function bits can easily be interpreted and modified. A pull-down menu helps to select the peripheral module you want to inspect.

TRACE32 supports simultaneous debug and trace of homogeneous and heterogeneous multicore and multiprocessor systems with one debug tool.

Start/stop synchronisation of the cores and a time-correlated display of the traced data gives you a global view of the system's state and the interplay of the cores.

**Multicore debugging**

**Core 0**

```

712 while ( ++count1 );
NSR:000022D0 E2820001
NSR:000022D4 E1B02000
NSR:000022D8 0A000000
NSR:000022DC EAFFFFF8
713
NSR:000022E0 E2811001
714
NSR:000022E4 EAFFFFF9
716
NSR:000022E8 E3A00001
NSR:000022EC E1A0F00E
          
```

**Core 1**

```

590 vtriplearray[0][0][0] = 1;
NST:000019FC 2001 movs r0,#0x1
NST:000019FE 4954 ldr r1,0x1B50
NST:00001A00 7008 strb r0,[r1]
591 vtriplearray[1][0][0] = 2;
NST:00001A02 2002 movs r0,#0x2
NST:00001A04 7308 strb r0,[r1,#0x0C]
592 vtriplearray[0][1][0] = 3;
NST:00001A06 2003 movs r0,#0x3
NST:00001A08 7108 strb r0,[r1,#0x4]
593 vtriplearray[0][0][1] = 4;
          
```

## OS-Aware Debugging and Tracing

Debugging of system and application software of all commonly used operating systems:

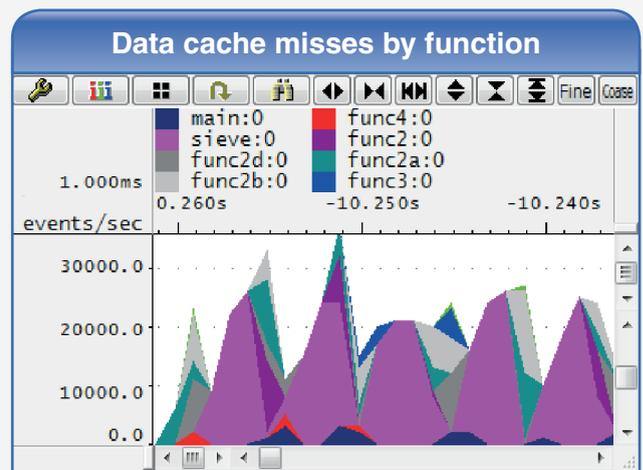
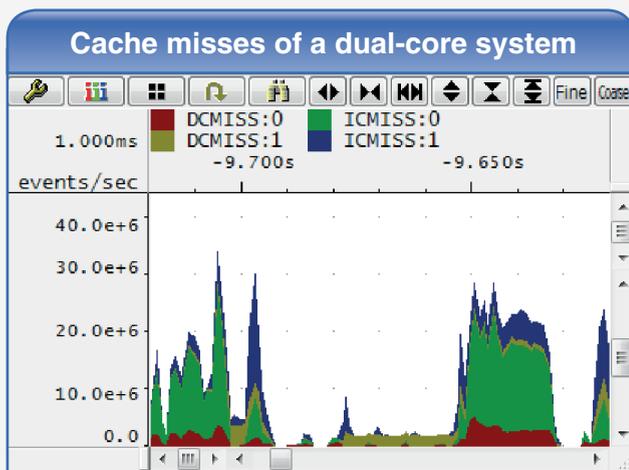
- Stop Mode Debugging: the whole system is stopped to analyse the overall system state
- Run Mode Debugging: the process of interest is stopped whilst kernel and other processes keep going
- Full support for symmetric (SMP) and asymmetric (AMP) multiprocessing
- Non-intrusive access to task lists and other kernel information
- Stack coverage and call hierarchy of waiting tasks
- Statistics and graphic display of task and function run-time

The screenshot shows a Linux debugger interface. On the left, a menu is open with 'Process Debugging' selected. A sub-menu is also open, listing options like 'Load Symbols...', 'Delete Symbols...', 'Debug Process on main...', 'Watch Processes', 'Scan Process MMU Pages...', and 'Scan All MMU Tables'. On the right, the 'List of processes' window displays a table of system processes.

magic	command	#thr	state	spaceid	pids
C08EA540	swapper/0	56.	running	0000	0. 2. 3. 4. 6. 7. 8. 9. 10.
EFFDFBC0	init	-	sleeping	0001	1.
EFC7DC00	ueventd	-	sleeping	0359	857.
EFC82960	sh	-	sleeping	04F3	1267.
EFC7F0A0	servicemanager	-	sleeping	04F4	1268.
EFF769E0	vold	3.	sleeping	04F5	1269. 1288. 1316.
EFC7F380	netd	7.	sleeping	04F6	1270. 1473. 1474. 1477. 147
EFC7F3A0	debuggerd	-	sleeping	04F7	1271.
	surfaceflinger	8.	running	04F8	1272. 1488. 1491. 1492. 149
	zygote	4.	sleeping	04F9	1273. 1880. 1881. 1882.
	drmserver	2.	sleeping	04FA	1274. 1483.
	mediaserver	5.	sleeping	04FB	1275. 1484. 1485. 1486. 154
	dbus-daemon	-	sleeping	04FC	1276.
	installd	-	sleeping	04FD	1277.
	keystore	-	sleeping	04FE	1278.
	uim	-	sleeping	0501	1281.
	addb	4.	sleeping	0502	1490. 1497. 1498. 1499.
	system_server	67.	current	05E4	1508. 1512. 1513. 1514. 151
	com.android.syst	10.	sleeping	0624	1572. 1576. 1577. 1578. 157
	android.process.	12.	sleeping	0633	1587. 1591. 1592. 1593. 159
ED423800	com.android.inpu	10.	sleeping	0641	1601. 1607. 1609. 1610. 161
ED425320	com.android.phon	21.	sleeping	0652	1618. 1623. 1624. 1625. 162
ED425320	com.android.laun	12.	sleeping	065C	1628. 1635. 1637. 1640. 164
EF0886C0	com.android.smsp	10.	sleeping	068C	1676. 1686. 1688. 1689. 169
ED5290A0	android.process.	15.	sleeping	0692	1682. 1691. 1692. 1694. 169
ED912900	com.android.desk	12.	sleeping	06BC	1724. 1726. 1728. 1730. 173
ED529940	com.android.prov	12.	sleeping	06D7	1751. 1753. 1754. 1756. 175
ECF36C60	com.android.exch	14.	sleeping	06EA	1770. 1774. 1775. 1776. 177

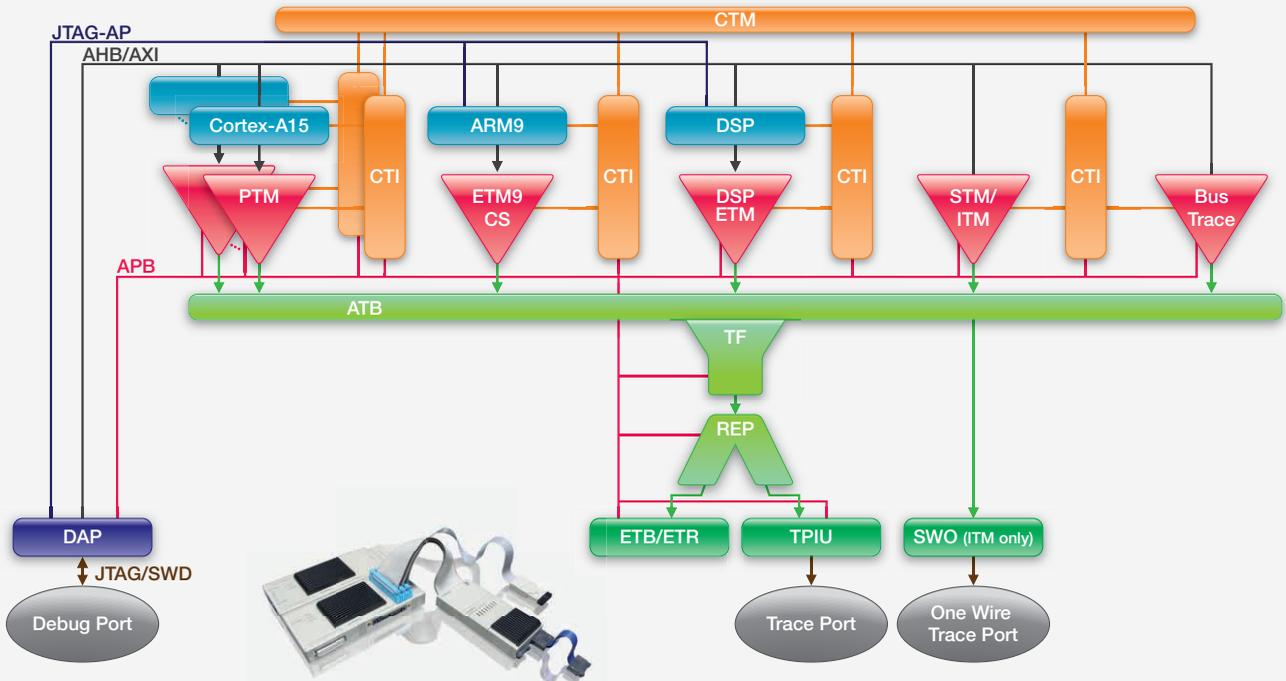
## Utilization of the Performance Monitors

Many ARM based devices include hardware counters with the ability to count specific hardware events like execution of certain kinds of instructions, cache or TLB events, or stall cycles caused by a specific reason. TRACE32 can sample these counters periodically. The results can be correlated with program trace information. This provides you with statistics on the system behaviour and system performance for finding bottlenecks and tuning the application.



## Full Support for ARM® CoreSight™ Technology

ARM CoreSight technology offers the chip designer various components to extend the core's debug functionality with the objective of debugging and tracing an entire system-on-chip. A joint JTAG or Serial Wire Debug interface allows real-time access to the on-chip buses and control of the cores and the CoreSight system itself. A common trace bus combines trace data from multiple sources like processor program and data trace, system trace information and accesses to the memory bus. TRACE32 displays time-correlated traces of multiple sources which had been stored in on-chip trace memory or emitted from a common trace port.



## Backward Debugging

Trace-based debugging allows developers to reconstruct the core context for any trace sampling point. You can re-debug a traced program section and watch how memory, registers and variables are changing. You can even step back in time and get a true high-level language trace listing showing register and stack variables.

**Forward and backward debugging capabilities**

addr/line	code	label	mnemonic	comment
SR:0000F6F4	035C0000		cmpeq r12,#0x0	; r12,#0
SR:0000F6F8	1A00000C		bne 0xF730	
186			/* AC terms all zero */	
SR:0000565C	E1B2C0F0		int dcval = DEQUANTIZE(inptr[DCTSIZ*0], c	

**Source level trace display showing local variables**

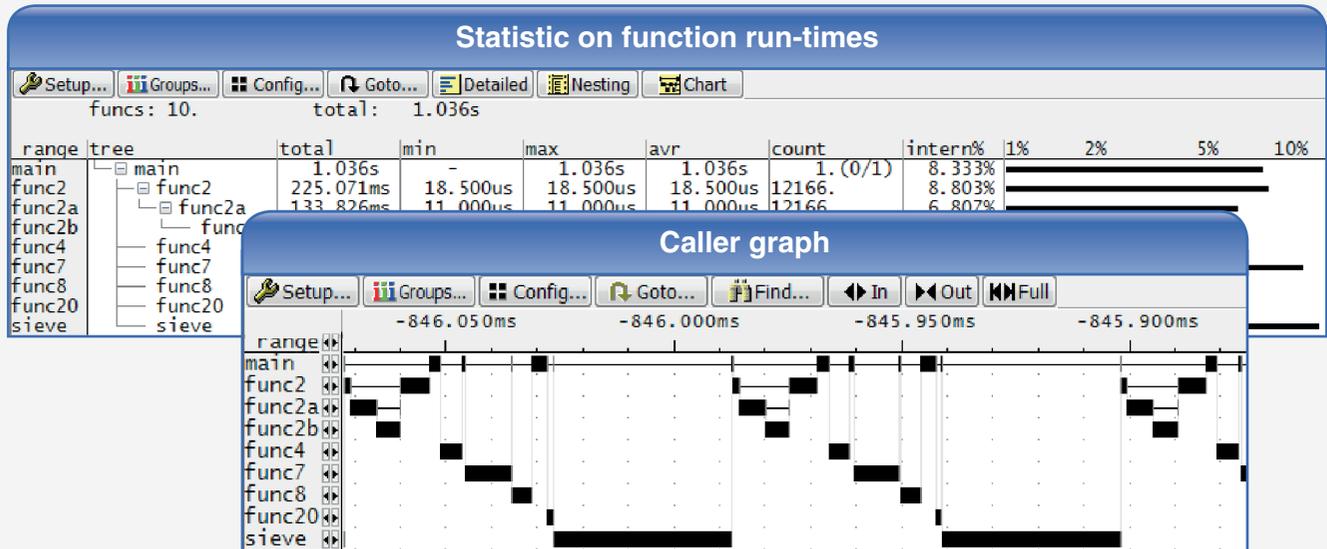
record	content
-000140	inptr = 0x0001E4A6 quantptr = 0x0001B934
231	tmp3 = DEQUANTIZE(inptr[DCTSIZ*1], quantptr[DCTSIZ*1]); tmp3 = -20
-000137	tmp0 = 0

The image shows a software interface for debugging. The top window displays assembly code with columns for address/line, code, label, mnemonic, and comment. The bottom window shows a source-level trace display with columns for record number and content. The content shows local variables like inptr, quantptr, tmp3, and tmp0. A yellow box highlights the Step, Over, and Entry buttons in the interface.

## Tracing and Profiling

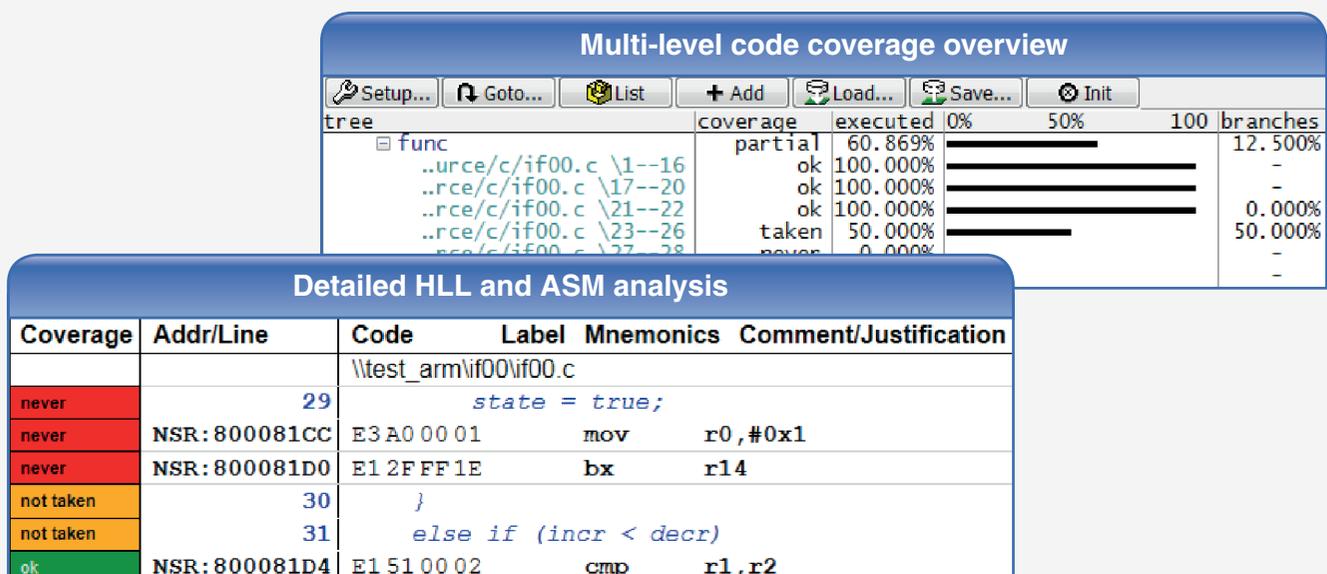
Powerful trace filter and trigger enable you to get the trace information of interest into the few kilobytes of on-chip trace memory or the up to 4 GB of TRACE32's off-chip trace memory. For long-term trace, streaming modes can be used which convey the trace data to the hard disk or to an application running on the host while recording.

This provides the capability of searching for bugs which only show up when running in real-time. In addition, there are various analysis functions, for example: run-time statistics on functions and tasks, analysis of the function nesting, or cache performance.



## Trace-based Code Coverage

An integrated real-time code coverage facility verifies statement and decision/branch coverage without prior code instrumentation. It is suitable to assist in the development of software for safety-related systems in compliance with ISO 26262, DO-178C, IEC 61508, and IEC 62304.





# ARM® support at a glance

More than 18 years of experience in ARM debugging enable us to provide the best-in-class debug and trace tools for ARM based systems:

- Multicore debugging and tracing for any mixture of ARM and DSP cores
- Support for all CoreSight components to debug and trace an entire SoC
- Powerful code coverage and run-time analysis of functions and tasks
- OS-aware debugging of kernel, libraries, tasks of all commonly used OSes



## Debugging ARM®-based Systems

The Lauterbach debug tools for ARM accompany you throughout the whole development process, from the early pre-silicon phase by debugging on an instruction set simulator or a virtual prototype, over board bring-up, till quality and maintenance work on the final product.

The features range from simple step/go/break, programming of on-chip flash, external NAND, eMMC, parallel and serial NOR flash devices, support for NEON and VFP units, to OS-aware debug and trace concepts for 32-bit and 64-bit multicore systems.

### Peripheral register

PMC (Power Management Controller)					
LVDSCL1	10	LVDF	Not detected	LVDAK NACK	LV DIE Disabled
		LV DRE	Enabled	LV DV Low trip	
LVDSCL2	20	LVVF	Not detected	LVWACK NACK	LVWIE Enabled
		LVVW	Low trip point		
REGSC	04	VLPRS	Not		
		REGONS	Run		

SD:4007D001 0--1 Low-Voltage Warning

The peripheral registers on the chip are displayed on a logical level. The function bits can easily be interpreted and modified. A pull-down menu helps to select the peripheral module you want to inspect.

TRACE32 supports simultaneous debug and trace of homogeneous and heterogeneous multicore and multiprocessor systems with one debug tool.

Start/stop synchronisation of the cores and a time-correlated display of the traced data gives you a global view of the system's state and the interplay of the cores.

### Multicore debugging

## OS-Aware Debugging and Tracing

Debugging of system and application software of all commonly used operating systems:

- Stop Mode Debugging: the whole system is stopped to analyse the overall system state
- Run Mode Debugging: the process of interest is stopped whilst kernel and other processes keep going
- Full support for symmetric (SMP) and asymmetric (AMP) multiprocessing
- Non-intrusive access to task lists and other kernel information
- Stack coverage and call hierarchy of waiting tasks
- Statistics and graphic display of task and function run-time

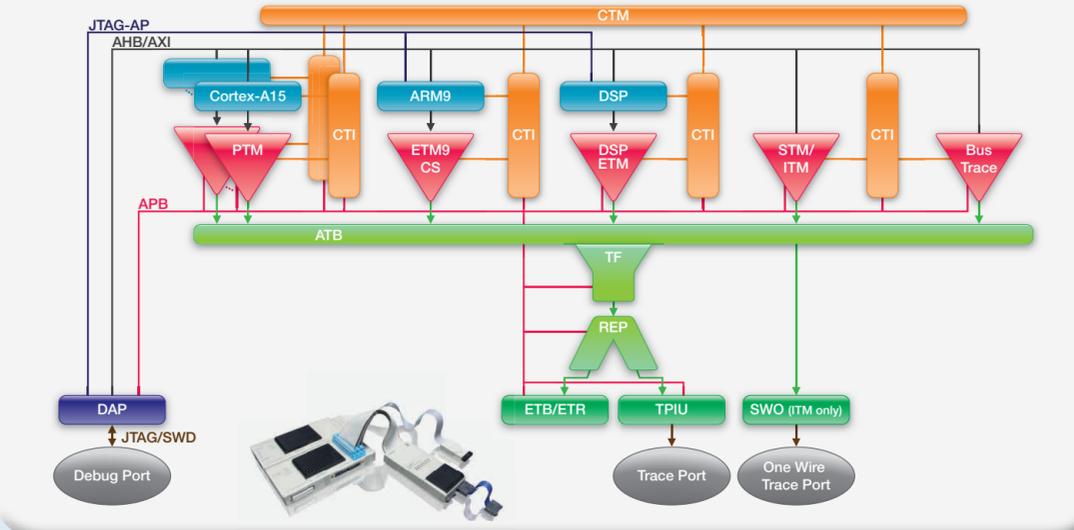
name	command	#thr	state	spaceid	pids
magic	swapper/0	56	running	0600	0. 2. 3. 4. 6. 7. 8. 9. 10. 11.
C08EA540	init	-	sleeping	0001	1.
EFFDFBC0	ueventd	-	sleeping	0359	857.
EFC7DC00	sh	-	sleeping	04F3	1268.
EFC82960	service manager	-	sleeping	04F4	1268.
EFC7F0A0	void	3.	sleeping	04F5	1269. 1288. 1316.
EFF769E0	netd	7.	sleeping	04F6	1270. 1473. 1474. 1477. 147
EFC7F380	debuggerd	-	sleeping	04F7	1271.
EFC7F380	surfaceflinger	8.	running	04F8	1272. 1488. 1491. 1492. 149
EFC7F380	zygote	4.	sleeping	04F9	1273. 1880. 1881. 1882.
EFC7F380	driver ver	2.	sleeping	04FA	1274. 1483.
EFC7F380	mediaserver	5.	sleeping	04FB	1275. 1484. 1485. 1486. 154
EFC7F380	dbus-daemon	-	sleeping	04FC	1276.
EFC7F380	installd	-	sleeping	04FD	1277.
EFC7F380	keystore	-	sleeping	04FE	1278.
EFC7F380	utim	-	sleeping	0501	1281.
EFC7F380	adbd	4.	sleeping	0502	1490. 1497. 1498. 1499.
EFC7F380	system_server	67.	current	05E4	1508. 1512. 1513. 1514. 151
EFC7F380	com.android.syst	10.	sleeping	0624	1572. 1576. 1577. 1578. 157
EFC7F380	android.process	12.	sleeping	0633	1587. 1591. 1592. 1593. 159
EFC7F380	com.android.inpu	10.	sleeping	0641	1601. 1607. 1609. 1610. 161
EFC7F380	com.android.phon	21.	sleeping	0652	1618. 1623. 1624. 1625. 162
EFC7F380	ED425320	12.	sleeping	065C	1628. 1635. 1637. 1640. 164
EFC7F380	ED425320	10.	sleeping	066C	1676. 1686. 1688. 1689. 169
EFC7F380	ED5290A0	15.	sleeping	0692	1682. 1691. 1692. 1694. 169
EFC7F380	ED912900	12.	sleeping	069C	1724. 1726. 1728. 1730. 173
EFC7F380	ED529940	12.	sleeping	06D7	1751. 1753. 1754. 1756. 175
EFC7F380	ECF36C60	14.	sleeping	06EA	1770. 1774. 1775. 1776. 177

## Utilization of the Performance Monitors

Many ARM based devices include hardware counters with the ability to count specific hardware events like execution of certain kinds of instructions, cache or TLB events, or stall cycles caused by a specific reason. TRACE32 can sample these counters periodically. The results can be correlated with program trace information. This provides you with statistics on the system behaviour and system performance for finding bottlenecks and tuning the application.

## Full Support for ARM® CoreSight™ Technology

ARM CoreSight technology offers the chip designer various components to extend the core's debug functionality with the objective of debugging and tracing an entire system-on-chip. A joint JTAG or Serial Wire Debug interface allows real-time access to the on-chip buses and control of the cores and the CoreSight system itself. A common trace bus combines trace data from multiple sources like processor program and data trace, system trace information and accesses to the memory bus. TRACE32 displays time-correlated traces of multiple sources which had been stored in on-chip trace memory or emitted from a common trace port.



## Backward Debugging

Trace-based debugging allows developers to reconstruct the core context for any trace sampling point. You can re-debug a traced program section and watch how memory, registers and variables are changing. You can even step back in time and get a true high-level language trace listing showing register and stack variables.

## Power Debug USB3

### Debugger

Debugging via JTAG or Serial Wire Debug interface.

Optional support for on-chip tracing (ETB, TMC).



### Debug and Trace for Small Trace Ports

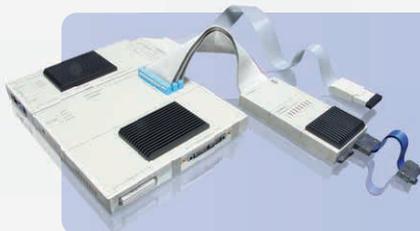
Optimized for, but not limited to Cortex-M based devices. It can capture an up to 4-bit wide Cortex-M or System Trace Port.

## Combi Probe

## Power Trace PX

### Debug and Trace

Debug and trace solution for parallel (TPIU) or serial (HSSTP) trace ports.



### High-End Debug and Trace

Debug and trace solution with large trace memory for fast parallel (TPIU) or serial (HSSTP) trace ports.

## Power Trace II

## µTrace®

### Debug and Trace for Cortex®-M

Cost-effective all-in-one debug and trace solution for Cortex-M based chips, only.



### Debugger for Virtual Targets

Debugging and tracing using TRACE32 on software models before a first hardware prototype is available.

## Front-End

For more information visit: [www.lauterbach.com/bdmarm.html](http://www.lauterbach.com/bdmarm.html)