LAUTERBACH
DEVELOPMENT TOOLS

TRACE32®

FASTEST DEBUG & TRACE
for Arm® Cortex®-M

μTrace®

Fastest Debug and Trace for Arm® Cortex®-M

µTrace® is Lauterbach's cost-effective all-in-one solution that provides the full feature set for Cortex-M microcontrollers implementing a compact trace port of up to 4 bits.

µTrace® provides the same features that Lauterbach's industry leading high-end products are known for: highest quality, exceptional functionality and superior support. In addition to its leading debug capabilities in the Cortex-M market, µTrace® can capture real-time information like system traces and parallel ETM flow traces, enabling e.g. code coverage and code profiling.

µTrace® supports data rates up to 400 Mbit/s per trace line from the CoreSight™ Trace Port Interface Unit (TPIU). This is 30% faster than competing trace products for Cortex-M based embedded systems. To ensure proper sampling at these speeds, the AutoFocus eye-finder automatically detects the perfect sampling point for each trace line.

µTrace® Characteristics

- Highest performance in the industry for Cortex-M (400 Mbit/s per trace line)
- Leading debug capabilities in the Cortex-M market
- Support for more than 7.000 different Cortex-M based chips
- USB 3 interface to the host computer
- Support for standard JTAG, Serial Wire Debug (SWD), and cJTAG (IEEE 1149.7)
- 256 MByte trace memory
- 10-/20-pin half-size connector for target hardware and a wide variety of adapters to work with other connectors
- Voltage range 1.2 to 5.0 V

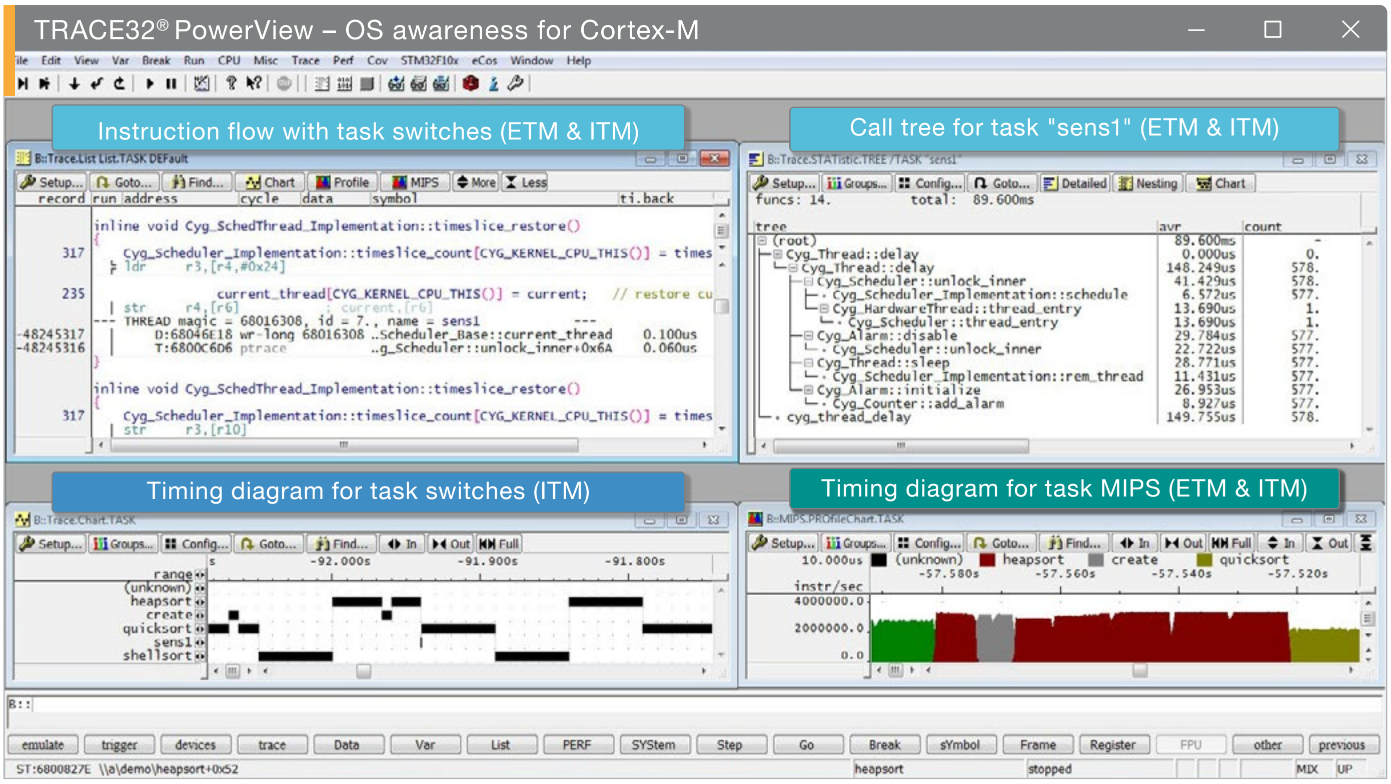
Debug Features

- C/C++ debugging
- Simple and complex breakpoints
- Read and write memory during program runs
- Flash programming support for FLASH memory on microcontrollers and external flash on the target system.
- Support for Armv8-M security extension (TrustZone®)
- OS-aware debugging: µTrace® allows for full-stack debugging of your target's operation system together with all tasks. The TRACE32 PowerView UI allows easy access to task lists and other OS specific information.

Selection of Supported Chip Families

Ambiq Micro	Apollo
AMD	Xilinx Artix-7
Analog Devices	ADuCM3, ADSP-CM4, MAX326xx
Arduino	Nicla Vision, Nano 33 BLE, Portenta H7
GigaDevice	GD32
Infineon	XMC1000/4000/7000, TLE98x, FM3, FM4, PSoC 4/5/6, Traveo II
Marvell	88Q5xxx, 88Q6xxx
Microchip	SmartFusion, SmartFusion2, ATSAM, PIC32
Nordic Semiconductor	nRF51, nRF52, nRF53, nRF91
Nuvoton	NuMicro
NXP	LPC8xx, LPC1xxx, Kinetis, JN518x, S32K, LPC4xxx, LPC5xxx, i.MX RT, QN90xx
onsemi	LC823450, RSL10
Raspberry Pi	RP2040
Renesas	RZ/G2L, DA14xxx, RA, R-IN32xx, ZSSC1856
Samsung	S3F
Silicon Labs	EFM32, EFR32, EM35
Socionext	SC1721/1722/1723
ST Microelectronics	STM32, Teseo V
Texas Instruments	Tiva C, Concerto, CC25xx, CC26xx, C32xx, MSP432
Toshiba	TX00, TX03, TX04, TXZ4

µTrace®


TRACE32® PowerView – OS awareness for Cortex-M
Instruction flow with task switches (ETM & ITM)
Call tree for task "sens1" (ETM & ITM)
Timing diagram for task switches (ITM)
Timing diagram for task MIPS (ETM & ITM)
B::Trace.List List.TASK DEFault
B::Trace.STATistic.TREE /TASK "sens1"
B::Trace.Chart.TASK
B::MIPS.PROfileChart.TASK

```
 record run address          cycle   data      symbol                          ti.back
           inline void Cyg_SchedThread_Implementation::timeslice_restore()
           {
      317     Cyg_Scheduler_Implementation::timeslice_count[CYG_KERNEL_CPU_THIS()] = times
            ldr      r3,[r4,#0x24]
      235                current_thread[CYG_KERNEL_CPU_THIS()] = current;    // restore cu
            | str      r4,[r6]             ; current,[r6]
           --- THREAD magic = 68016308, id = 7., name = sens1             ---
-48245317   |    D:68046E18 wr-long 68016308 ..Scheduler_Base::current_thread     0.100us
-48245316   |    T:6800C6D6 ptrace           ..g_Scheduler::unlock_inner+0x6A     0.060us
           }
           inline void Cyg_SchedThread_Implementation::timeslice_restore()
           {
      317     Cyg_Scheduler_Implementation::timeslice_count[CYG_KERNEL_CPU_THIS()] = times
            | str      r3,[r10]
```

```
funcs: 14.          total:   89.600ms

tree                                                   avr         count
⊟ (root)                                               89.600ms        -
├⊟ Cyg_Thread::delay                                   0.000us        0.
└⊟ Cyg_Thread::delay                                 148.249us      578.
  ├⊟ Cyg_Scheduler::unlock_inner                      41.429us      578.
  │├ · Cyg_Scheduler_Implementation::schedule          6.572us      577.
  │└⊟ Cyg_HardwareThread::thread_entry                13.690us        1.
  │  └ · Cyg_Scheduler::thread_entry                  13.690us        1.
  ├⊟ Cyg_Alarm::disable                               29.784us      577.
  │└ · Cyg_Scheduler::unlock_inner                    22.722us      577.
  ├⊟ Cyg_Thread::sleep                                28.771us      577.
  │└ · Cyg_Scheduler_Implementation::rem_thread       11.431us      577.
  └⊟ Cyg_Alarm::initialize                            26.953us      577.
    └ · Cyg_Counter::add_alarm                         8.927us      577.
└ · cyg_thread_delay                                 149.755us      578.
```

▲ **FIG. 1:** Through the combination of ETM and ITM trace data, extensive trace analysis can be provided for several operating systems.

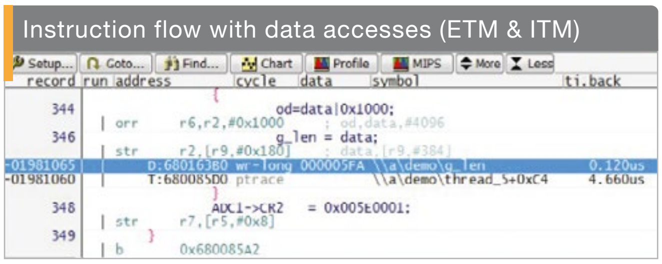
Trace Features

- Support of program-flow traces created by the ETM
- Support of data and event-traces created by the ITM or ETMv4.x
- Off-chip trace of ETM/ITM via TPIU in continuous mode with up to 4 trace pins for detailed analysis
- Off-chip trace of ITM via Serial Wire Output (SWO) for SoCs without TPIU
- On-chip trace of ETM/ITM via ETB (if available) for SoCs without TPIU
- On-chip trace of program-flow via MTB for SoCs without ETM
- Combining of ETM and ITM trace data for a seamless integration of read/write accesses into the instruction flow
- OS-aware tracing
- Analysis of task and function run times
- Trace the execution state of all branch and non-branch instructions
- Code coverage analysis
- On-the-fly transferring of trace data to the host computer
- Energy measurement using optional TRACE32® Mixed-Signal Probe
- Concurrent recording of 12 digital signals with optional TRACE32® Mixed-Signal Probe

Combining ETM and ITM Trace Data

The ETM generates information about the instructions that have been executed. The ITM generates information about the read/write accesses assisted by the Data Watchpoint and Trace unit (DWT).

The ITM trace packets for read/write accesses contain the following information: data address, data value, program counter. Through analysis of the program counter, these separately generated data accesses can be seamlessly integrated into the instruction flow (*figure 2*). This in turn leads to significantly faster error location. The cause of an error such as an incorrect data value being written to an address can be found easily if the write accesses are embedded into the overall instruction flow.



```
 record run address          cycle   data      symbol                          ti.back
                       {
      344                        od=data|0x1000;
          | orr     r6,r2,#0x1000      ; od,data,#4096
      346                        g_len = data;
          | str     r2,[r9,#0x180]     ; data,[r9,#384]
-01981065 |     D:680163B0 wr-long 000005FA \\a\demo\g_len               0.120us
-01981060 |     T:680085D0 ptrace           \\a\demo\thread_5+0xC4       4.660us
                       }
      348              ADC1->CR2    = 0x005E0001;
          | str     r7,[r5,#0x8]
      349         }
          | b       0x680085A2
```

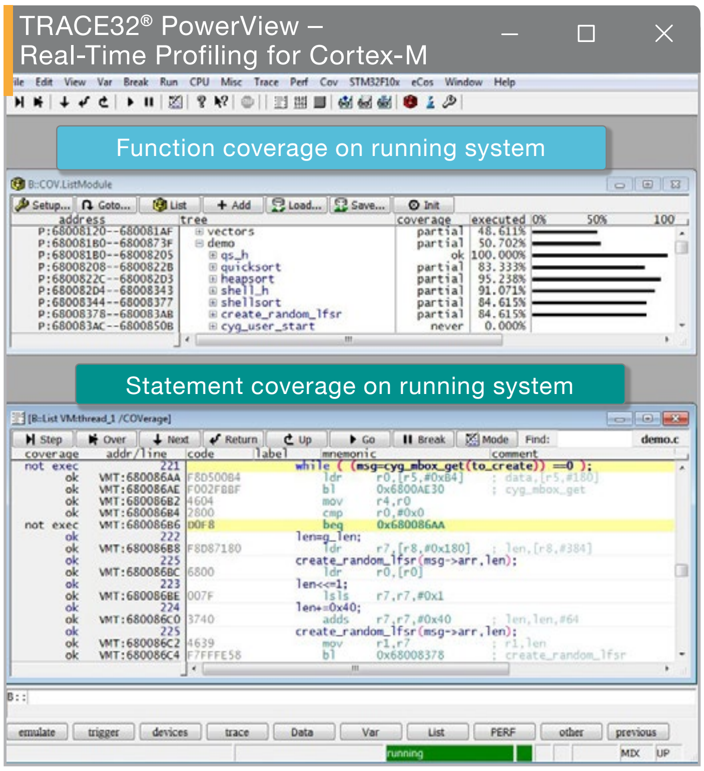
▲ **FIG. 2:** By combining ETM and ITM trace data, read/write accesses can be integrated seamlessly into the instruction flow.

OS-Aware Tracing

If an operating system is running, task switch information becomes essential for the trace analysis.

The ITM can generate trace information for each task-switch. Therefore it emits a data-trace message whenever the kernel writes on the variable indicating the active task.

As described on the previous page, data-trace messages can be integrated seamlessly into the instruction flow trace. As a result the task switch information is also integrated into the instruction trace. This improves the readability of the trace listing. The integration of the task switch into the instruction flow trace also forms the basis for the run-time analysis as shown in figure 1.


TRACE32® PowerView –
Real-Time Profiling for Cortex-M
ile Edit View Var Break Run CPU Misc Trace Perf Cov STM32F10x eCos Window Help
Function coverage on running system
Setup... Goto... List Add Load... Save... Init
Statement coverage on running system
Step Over Next Return Up Go Break Mode Find: demo.c
emulate trigger devices trace Data Var List PERF other previous
running MIX UP

```
B::COV.ListModule
     address             tree                   coverage  executed 0%   50%   100
 P:68008120--680081AF    ⊞ vectors              partial    48.611%
 P:680081B0--6800873F    ⊟ demo                 partial    50.702%
 P:680081B0--68008205      ⊞ qs_h                    ok   100.000%
 P:68008208--6800822B      ⊞ quicksort          partial    83.333%
 P:6800822C--680082D3      ⊞ heapsort           partial    95.238%
 P:680082D4--68008343      ⊞ shell_h            partial    91.071%
 P:68008344--68008377      ⊞ shellsort          partial    84.615%
 P:68008378--680083AB      ⊞ create_random_lfsr partial    84.615%
 P:680083AC--6800850B      ⊞ cyg_user_start       never     0.000%
```

```
[B::List VMthread_1 /COVerage]
coverage   addr/line      code      label  mnemonic                         comment
not exec            221                    while ( (msg=cyg_mbox_get(to_create)) ==0 );
      ok  VMT:680086AA  F8D500B4             ldr     r0,[r5,#0xB4]         ; data,[r5,#180]
      ok  VMT:680086AE  F002FBBF             bl      0x6800AE30            ; cyg_mbox_get
      ok  VMT:680086B2  4604                 mov     r4,r0
      ok  VMT:680086B4  2800                 cmp     r0,#0x0
not exec  VMT:680086B6  D0F8                 beq     0x680086AA
      ok            222                    len=g_len;
      ok  VMT:680086B8  F8D87180             ldr     r7,[r8,#0x180]        ; len,[r8,#384]
      ok            225                    create_random_lfsr(msg->arr,len);
      ok  VMT:680086BC  6800                 ldr     r0,[r0]
      ok            223                    len<<=1;
      ok  VMT:680086BE  007F                 lsls    r7,r7,#0x1
      ok            224                    len+=0x40;
      ok  VMT:680086C0  3740                 adds    r7,r7,#0x40           ; len,len,#64
      ok            225                    create_random_lfsr(msg->arr,len);
      ok  VMT:680086C2  4639                 mov     r1,r7                 ; r1,len
      ok  VMT:680086C4  F7FFFE58             bl      0x68008378            ; create_random_lfsr
B::
```

▲ **FIG. 3:** Real-time Profiling enables code coverage analysis to be followed live on the screen.

Recording Modes

To record the trace information generated by the ETM and ITM, µTrace® supports various modes:

> FIFO AND STACK MODE

The complete trace information is stored inside the 256 MByte memory of the TRACE32® µTrace®.

> STREAM MODE

The complete trace information is streamed to the host computer and stored in a file on its hard-disk.

> REAL-TIME PROFILING

The trace information relevant for the selected profiling is streamed to the host computer and analyzed during runtime.

Each recording mode has its specific strengths.

FIFO and STACK Mode are the most commonly used modes. In STACK mode trace recording stops when the trace memory is full. FIFO mode overwrites the trace memory until the recording stops. The trace analysis is then undertaken after recording is completed. These modes work quickly (200 MByte/s) and are sufficient for error location and run-time analysis.

STREAM Mode can collect trace data for a long time period. This is helpful if an error location cannot be triggered, if documentation of a program run is required over a long time or if you want to perform a statistical analysis over a longer period. In stream mode µTrace® supports an average trace-rate of 140 MByte/s with temporary peaks up to 200 MByte/s.

Real-time Profiling is particularly suitable for performing object statement and object branch coverage. The coverage analysis can be followed live on the screen and the test results are visible immediately (*see Figure 3*). Lines marked as "ok" have already been covered, lines marked as "not exec" may require additional test stimuli.

FOR MORE INFORMATION VISIT: www.lauterbach.com/microtrace
www.lauterbach.com/os-awareness