NEWS 2009

LAUTERBACH
DEVELOPMENT TOOLS

DEBUGGER, REAL-TIME TRACE, LOGIC ANALYZER

THE NEW HEADQUARTERS IN GERMANY

30 Years Lauterbach – More Space for Creative Minds

Lauterbach continues their 30 year success story and, in December 2008, moved its corporate headquarters into the Arcone Technology Center. This more ergonomic space facilitates the intention to expand our team; contributing to even better customer service by taking the innovative character of TRACE32 products and our superb technical support to a higher level.

What started as a one-man show in 1979 has now grown into a company that operates worldwide, with 70 employees in Germany and 30 more working at branches around the globe. The move to larger premises was absolutely essential for Lauterbach to be able to continue growing over the next few years.

The Arcone Technology Center

The Arcone Technology Center in Höhenkirchen-Siegertsbrunn near Munich was built by Lauterbach in two years – designed as office premises to house Lauterbach as well as other medium-sized technology companies. Its close proximity to Munich and the airport was a decisive factor in the choice of Höhenkirchen-Siegertsbrunn as the new company headquarters. The locality provides easy access to various German autobahns and good public transport connections.

The Arcone Technology Center provides plenty of space for Lauterbach staff. The flexible workplaces can be arranged to suit the tasks and needs of the individual. There are many shared open spaces to encourage the productive exchange of innovative ideas.

Since Lauterbach plans to extend its range of vocational and other training courses it was important that the building would provide suitably designed spaces, this need was included during the planning phase. »

CONTENTS

For our customer trainings, conferences, and other events, various purpose-built rooms are available, all equipped with the latest media technology. The Arcone Technology Center houses a separate restaurant that offers our visitors an organic menu.

The New Corporate Design

The move has also provided Lauterbach with the opportunity to give its corporate design a makeover. This update and all related activities will be concluded by the end of March 2009. We are presenting this new design on our exhibition booth at *Embedded Systems Conference Silicon Valley 2009* in San Jose.

Fig. 1: Shared open spaces for planned or impromptu meetings designed to encourage the exchange of innovative ideas

We would like to show you our latest developments on the following pages of our NEWS 2009. Many of these will be on display and demonstrated live at *ESC 2009*. We will be happy to greet you at our booth 1532.

Floating Licenses for TRACE32 Front-End

Since September 2008, Lauterbach has been offering floating licenses for its TRACE32 Front-End product range.

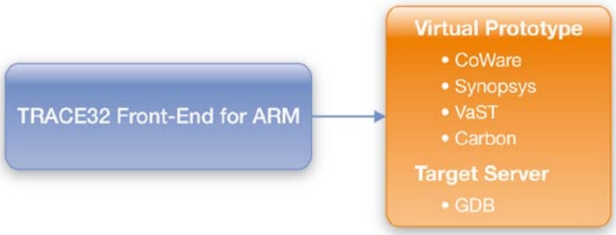


Fig. 2: The "TRACE32 Front-End for ARM" supports various virtual prototypes and GDB

In addition to traditional debugging on the target hardware, we now find that virtual prototypes are often used to test embedded software. Instead of the JTAG interface, a software interface (API) provides the interface to the debugging functions. Lauterbach now provides its debug software frontend so that customers can debug using the sophisticated TRACE32 GUI.

The software frontend is designed so that beside virtual prototypes also other software solutions are supported for debugging. This includes core simulators provided by the semiconductor manufacturers as well as target servers such as GDB.

Lauterbach offers a "TRACE32 Front-End" for every common embedded processor architecture: as an example on the ARM and Cortex processors we provide the "TRACE32 Front-End for ARM". All versions are supplied with the appropriate current debug APIs (see Figure 2).

There are two license models for the use of a TRACE32 frontend:

1. Floating licenses

With the Reprise License Manager RLM, a specific number of frontend licenses can be made available in a network. For more information about RLM, see: www.reprisesoftware.com.

2. USB Dongle

For single-user licenses, a USB dongle is available.

LONG-TERM TRACE ETMv3

Implementation

Lauterbach will present the long-term trace for the ARM ETMv3 at Embedded Systems Conference Silicon Valley 2009. The aim of this innovation is to enable greatly extended measurement times for TRACE32 profiling and code-coverage functions.

This article describes the concepts of the long-term tracing technology as well as the technical requirements it places on the trace tool and the respective host computer.

ARM ETMv3

Tracing means recording detailed information about a program as it runs on the core. This information is usually generated by on-chip trace logic. For ARM cores, this logic element is known as the *Embedded Trace Macrocell* or ETM. The latest version of this logic, the ETMv3, can be found today in most ARM11 and Cortex cores. Since the functionality of the on-chip trace logic is the basis for the trace data, let's start with a brief introduction.

The ETMv3 generates a package-oriented trace log. The following information can be generated at program runtime and collected in trace packages:

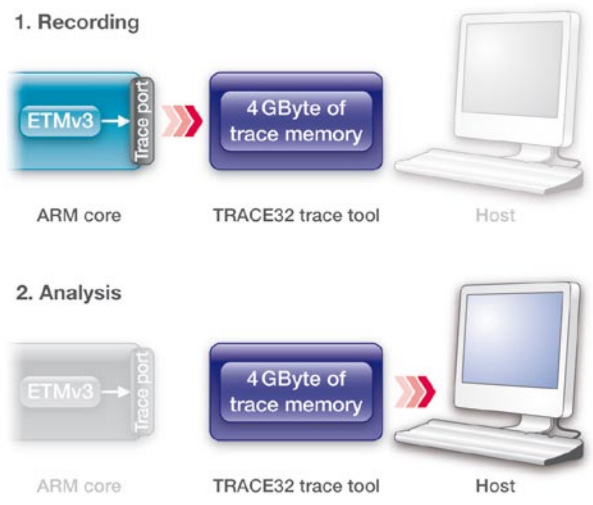
- **Program flow packages:** Contain information about the program instructions executed by the core – mainly the target addresses of jumps as well as the number of instructions executed between two jumps.
- **Data flow packages:** Contain the memory addresses read/written by the program as well as the respective data values.
- **Context-ID packages:** Contain a process/task ID in the event that an operating system is running.

The trace packages are output by the on-chip trace logic via the trace port. The trace port for the ETMv3 typically consists of 8 or 16 pins for the trace packages plus two pins for the control signals.

To minimize the bandwidth for the package output, the ETMv3 compresses the trace packages. For example, all addresses are shortened by a special algorithm. However, if the data volume is higher than the maximum bandwidth of the trace port, the FIFO buffer can overflow and some trace packages can be lost.

Compressing the trace information alone is not enough to prevent FIFO overflows. The next stage is provided by the programmability of the ETMv3. To reduce the number of trace packages, you can simply define what trace information you want generated and output. For example, no data flow information is needed for the TRACE32 profiling functions. This is very useful because it is mainly the data packages that inflate the trace volume at the port.

Classic Tracing



Classic tracing means: record first, then analyze

Classical tracing currently consists of two steps that are carried out consecutively:

»

1. Recording

The trace packages are sampled at the trace port and placed in the trace memory.

2. Analysis

The trace packages are transferred from the trace memory to the host, where they are decompressed and analyzed.

The classical tracing method is limited in that only the section of the program that fits within the buffer memory can be saved for evaluation and analysis. The memory depth of the TRACE32 trace tools is currently between 1 GBytes and 4 GBytes. This allows the recording of up to 3 G trace packages.

Recording

In classical tracing, the real challenge is the recording of the trace packages. Since ARM cores today usually run at frequencies up to 1 GHz, only a fast trace port can guarantee the loss-free output of all trace packages.

The Lauterbach trace tools for the parallel ETMv3 support package recording at a frequency of up to 275 MHz DDR and can therefore handle the following data rates (see Figure 3):

- 8.8 GBit/s with 16 pins for the trace package
- 4.4 GBit/s with 8 pins for the trace packages

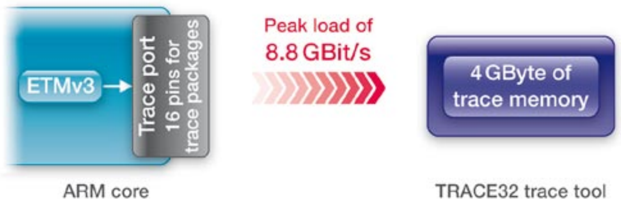


Fig. 3: The trace tool for the parallel ETMv3 supports a data rate of 8.8 GBit/s with 16 pins for the trace packages

With the serial trace tools for the ETMv3, data rates of up to 20 GBit/s can be recorded. For detailed information on serial trace tools, see page 10.

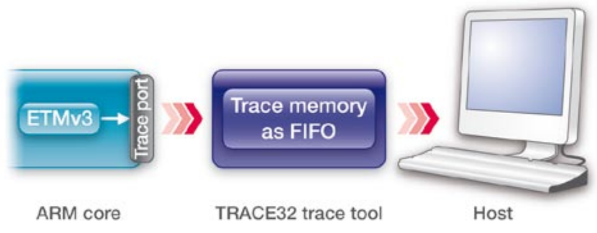
Analysis

To analyze the recorded program section, you have to transfer the trace packages from the trace memory to the host, decompress and then evaluate them.

Since the trace packages for the program flow contain no program code, this has to be added prior to analysis. The following data is used:

- The program code read by the TRACE32 software from the target system memory over the JTAG interface.
- Symbol and debug information loaded by the user for the TRACE32 software.

Long-Term Tracing



Long-term tracing means: recording and direct analysis

Long-term tracing is implemented by transferring the trace packages to the host during recording and analyzing them immediately. In this case, the trace memory of the TRACE32 trace tool is basically used as just a FIFO.

Extremely large data volumes are created during a long-term trace, so it is advisable to analyze the trace packages in parallel with the recording. Even if the trace packages are compressed before they are stored in a file, up to 5 GBytes are typically collected per hour. At the same time, you have to allow a lot of time for further analysis after recording is completed. For example, if you collect trace packages for a two-hour program run in a file, you need several hours for a subsequent conventional analysis, even on a high-powered host.

»

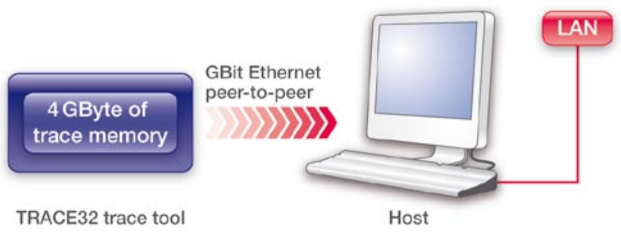


Fig. 4: Long-term tracing needs a fast peer-to-peer interface to the host

If large data volumes are to be quickly recorded, transferred, and analyzed in long-term tracing, the following conditions must be met:

- Fast host
- Fast trace tool
- Compact data formats

Fast Host

In order to be able to analyze the trace packages on the host at program runtime, you need a fast dual-core computer. Here, one core receives the trace packages while the second core evaluates the packages in parallel.

For the analysis, the program code is needed in addition to the trace packages. Since many ARM cores are not able to read the code from the target system memory at runtime, the code must be copied to the TRACE32 software before the start of the long-term trace.

Fast Trace Tool

As described above for classical tracing, the trace tool has to sample the trace packages loss-free at a fast trace port. High-speed transfer of the trace packages to the host is the new requirement for long-time tracing. For this purpose, the TRACE32 trace tool provides a GBit Ethernet interface. If the trace tool is connected peer-to-peer to the host, a transmission rate of more than 500 MBit/s can be achieved (see Figure 4).

The maximum transmission rate to the host is currently the bottleneck of long-term tracing. This means that long-term tracing will only work if the average data rate at the trace port does not exceed the maximum transmission rate to the host (see Figure 5).

High peak loads are not critical since they can be buffered by the trace memory.

»

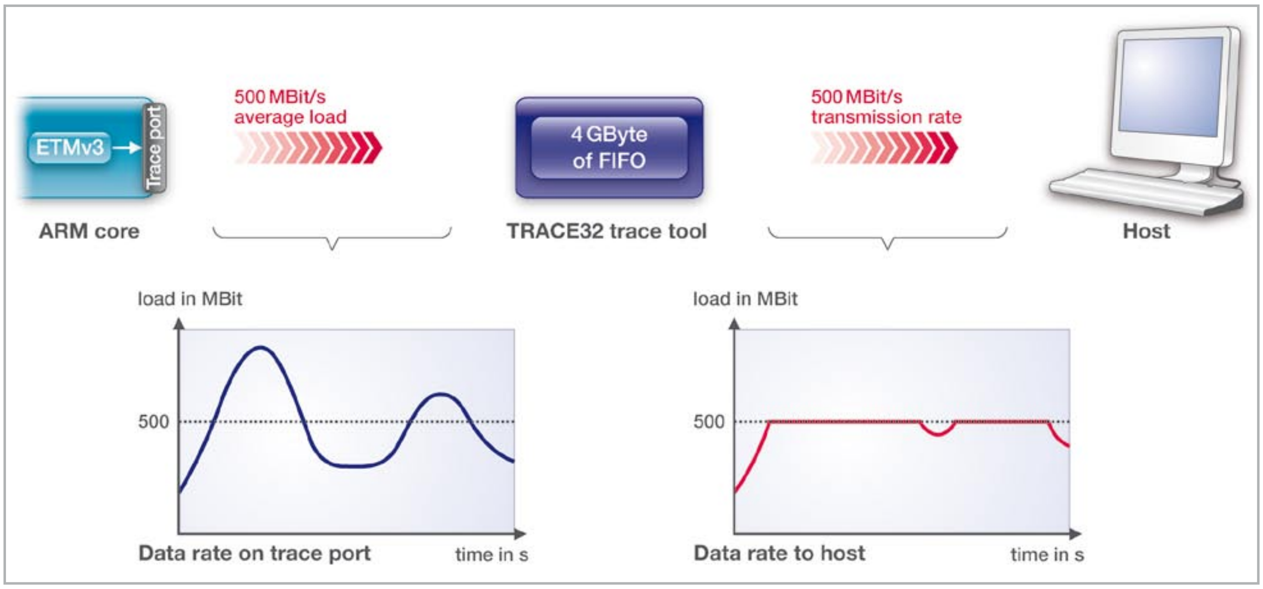


Fig. 5: Long-term tracing works for this example if the average load at the trace port does not exceed 500 MBit/s

Software	Mobile Terminal	Floating Point Arithmetic	HDD Controller
Trace information per instruction	0.8 Bit	2.2 Bit	4.3 Bit
Core	Cortex-A	ARM11	ARM9
Core frequency	500 MHz	300 MHz	450 MHz
Trace port frequency DDR	166 MHz	75 MHz	150 MHz
RTOS	Linux	–	–
Average data rate at trace port	340 MBit/s	406 MBit/s	798 MBit/s

Compact Data Formats

Since the maximum transmission rate to the host is limited, it is important to keep the data volume as compact as possible. The data volume can be influenced in two ways:

1. Optimal programming of ETMv3
2. Compact buffering of trace packages

1. Optimal programming of ETMv3

You can directly influence the data rate at the trace port by programming the ETMv3 so that trace packages are only generated for analysis-relevant information. The data flow packages that represent a high load for the trace port are not usually needed for profiling and code coverage.

The other factors influencing the average data rate at the trace port unfortunately have to be considered as unchangeable:

ARM core frequency: The higher the ARM core frequency, the more trace data per second.

Software on the target system: A software program that makes a large number of jumps and finds data/instructions in the cache generates more trace packages per second than a software program that pro- »

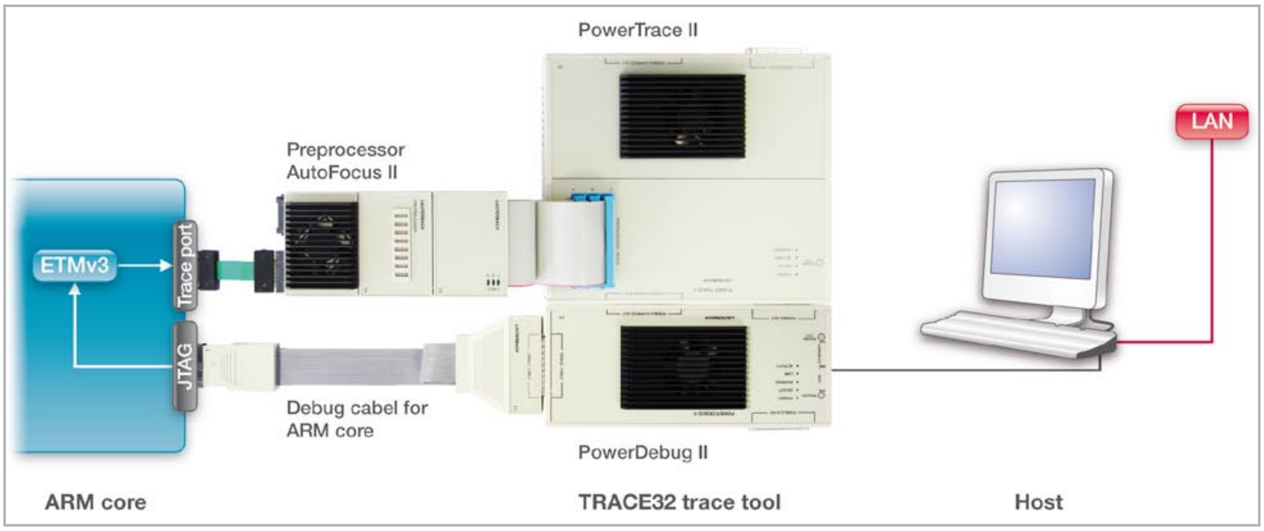


Fig. 6: The Lauterbach trace tool for long-term tracing with the ETMv3

cesses many sequential instructions and often has to wait for the availability of data/instructions.

The table on page 6 shows a few examples of average data rate measurements at the trace port. It is surprising that the data rate is greatly influenced by the software running on the core. The core frequency and the core architecture do not play as significant of a role.

2. Compact buffering

The firmware of the TRACE32 trace tool has been enhanced so that, the optimal packing density of the packages in the trace memory is achieved with 8 pins for the output of the trace packages.

Summary

In the TRACE32 software, the configuration and analysis of the long-term trace runs under the name of *Real-Time Streaming* – RTS for short.

A Lauterbach trace tool for long-term tracing of the parallel ETMv3 consists of the following TRACE32 products (see Figure 6):

PowerDebug II: Provides the GBit Ethernet interface to the host and transmits the trace packages.

Debug cable for the ARM core: Programs the ETMv3 over the JTAG interface.

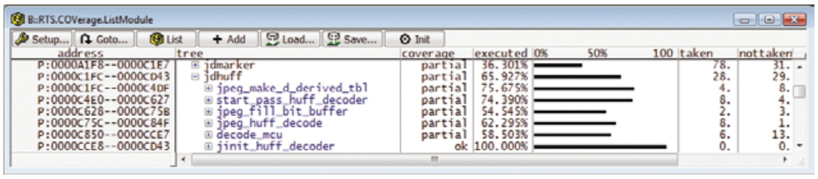
PowerTrace II: Stores the trace packages, max. trace depth currently 4 GBytes.

Preprocessor AutoFocus II: Samples the trace packages at the parallel trace port and transfers them to the trace memory.

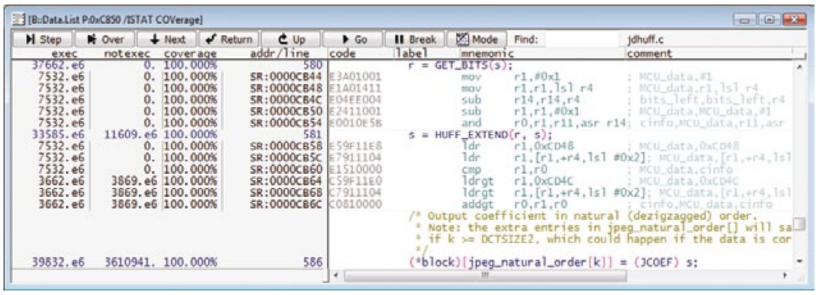
With long-term tracing, Lauterbach has taken an important step toward a trace technology that permits an almost unlimited analysis of the program run. It is fair to assume that both the processing power of the hosts and hard-disk capacity will increase constantly during the next few years, so we expect even more comprehensive analysis options will become available.

LONG-TERM TRACE ETMv3

Code Coverage-Analysis and Long-Term Tracing



```
address                  tree                          coverage  executed 0%    50%    100 taken  nottaken
P:0000A1F8--0000C1E7     ⊞ jdmarker                     partial   36.301%                    78.      31.
P:0000C1FC--0000CD43     ⊟ jdhuff                       partial   65.927%                    28.      29.
P:0000C1FC--0000C4DF       ⊞ jpeg_make_d_derived_tbl    partial   75.675%                     4.       8.
P:0000C4E0--0000C627       ⊞ start_pass_huff_decoder    partial   74.390%                     8.       4.
P:0000C628--0000C75B       ⊞ jpeg_fill_bit_buffer       partial   54.545%                     2.       3.
P:0000C75C--0000C84F       ⊞ jpeg_huff_decode           partial   62.295%                     8.       1.
P:0000C850--0000CCE7       ⊞ decode_mcu                 partial   58.503%                     6.      13.
P:0000CCE8--0000CD43       ⊞ jinit_huff_decoder              ok  100.000%                     0.       0.
```



```
     exec   notexec  coverage    addr/line code      label  mnemonic                       comment
 37662.e6        0.  100.000%          580                  r = GET_BITS(s);
  7532.e6        0.  100.000% SR:0000CB44 E3A01001             mov    r1,#0x1               ; MCU_data,#1
  7532.e6        0.  100.000% SR:0000CB48 E1A01411             mov    r1,r1,lsl r4          ; MCU_data,r1,lsl r4
  7532.e6        0.  100.000% SR:0000CB4C E04EE004             sub    r14,r14,r4            ; bits_left,bits_left,r4
  7532.e6        0.  100.000% SR:0000CB50 E2411001             sub    r1,r1,#0x1            ; MCU_data,MCU_data,#1
  7532.e6        0.  100.000% SR:0000CB54 E0010E58             and    r0,r1,r11,asr r14; cinfo,MCU_data,r11,asr
 33585.e6  11609.e6  100.000%          581                  s = HUFF_EXTEND(r, s);
  7532.e6        0.  100.000% SR:0000CB58 E59F11E8             ldr    r1,0xCD48             ; MCU_data,0xCD48
  7532.e6        0.  100.000% SR:0000CB5C E7911104             ldr    r1,[r1,+r4,lsl #0x2]; MCU_data,[r1,+r4,lsl
  7532.e6        0.  100.000% SR:0000CB60 E1510000             cmp    r1,r0                 ; MCU_data,cinfo
  3662.e6   3869.e6  100.000% SR:0000CB64 C59F11E0             ldrgt  r1,0xCD4C             ; MCU_data,0xCD4C
  3662.e6   3869.e6  100.000% SR:0000CB68 C7911104             ldrgt  r1,[r1,+r4,lsl #0x2]; MCU_data,[r1,+r4,lsl
  3662.e6   3869.e6  100.000% SR:0000CB6C C0810000             addgt  r0,r1,r0              ; cinfo,MCU_data,cinfo
                                                            /* Output coefficient in natural (dezigzagged) order.
                                                             * Note: the extra entries in jpeg_natural_order[] will sa
                                                             * if k >= DCTSIZE2, which could happen if the data is cor
                                                             */
 39832.e6  3610941.  100.000%          586                  (*block)[jpeg_natural_order[k]] = (JCOEF) s;
```

Fig. 7: Lists showing code coverage (overview and detailed)

One application for long-term tracing is checking whether all of the program code is processed during a system test.

For this code-coverage analysis of the trace data, the TRACE32 software provides a list of all modules/functions and their code coverage. Additionally, a statistical summary of the execution of conditional instructions is displayed. Each function can be analyzed in detail: For linear code, you can see how often a command was run during the test (exec). For conditional instructions, you can also observe how often a command was skipped because its condition was not met (notexec).

LONG-TERM TRACE ETMv3

Profiling and Long-Term Tracing

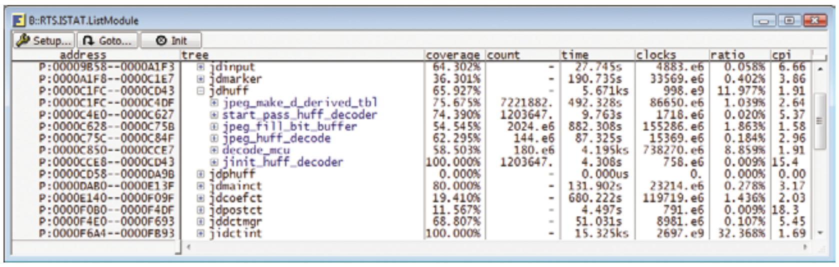
For time-critical functions, maximum times are often defined and have to be checked during the system test. The long-term tracing feature makes this check easy and causes of timeouts to be quickly found.

Verification

First is the check whether time-critical functions exceed their maximum time. For this purpose, the ETMv3 should be programmed so that it generates only trace packages for the program flow and the context ID. There are two reasons for this:

1. In this way, the data rate at the trace port is kept as low as possible.

2. FIFO overflows preventing an exact analysis of function nesting are prohibited.

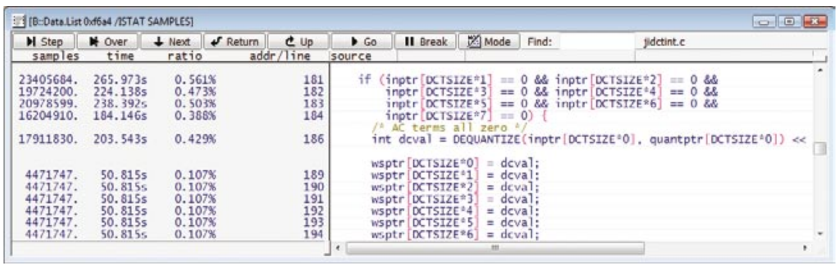


```
B::RTS.ISTAT.ListModule
Setup...  Goto...  Init
       address            tree                             coverage  count      time        clocks     ratio    cpi
  P:00009B58--0000A1F3    ⊞ jdinput                          64.302%          -     27.745s    4883.e6    0.058%  6.66
  P:0000A1F8--0000C1E7    ⊞ jdmarker                         36.301%          -    190.735s   33569.e6    0.402%  3.86
  P:0000C1FC--0000CD43    ⊟ jdhuff                           65.927%          -     5.671ks     998.e9   11.977%  1.91
  P:0000C1FC--0000C4DF      ⊞ jpeg_make_d_derived_tbl        75.675%   7221882.    492.328s   86650.e6    1.039%  2.64
  P:0000C4E0--0000C627      ⊞ start_pass_huff_decoder        74.390%   1203647.      9.763s    1718.e6    0.020%  5.37
  P:0000C628--0000C75B      ⊞ jpeg_fill_bit_buffer           54.545%    2024.e6    882.308s  155286.e6    1.863%  1.58
  P:0000C75C--0000C84F      ⊞ jpeg_huff_decode               62.295%     144.e6     87.325s   15369.e6    0.184%  2.96
  P:0000C850--0000CCE7      ⊞ decode_mcu                     58.503%     180.e6     4.195ks  738270.e6    8.859%  1.91
  P:0000CCE8--0000CD43      ⊞ jinit_huff_decoder            100.000%   1203647.      4.308s     758.e6    0.009% 15.4
  P:0000CD58--0000DA9B    ⊞ jdphuff                           0.000%          -     0.000us         0.    0.000%  0.00
  P:0000DAB0--0000E13F    ⊞ jdmainct                         80.000%          -    131.902s   23214.e6    0.278%  3.17
  P:0000E140--0000F09F    ⊞ jdcoefct                         19.410%          -    680.222s  119719.e6    1.436%  2.03
  P:0000F0B0--0000F4DF    ⊞ jdpostct                         11.567%          -      4.497s     791.e6    0.009% 18.3
  P:0000F4E0--0000F693    ⊞ jddctmgr                         68.807%          -     51.031s    8981.e6    0.107%  5.45
  P:0000F6A4--0000FB93    ⊞ jidctint                        100.000%          -    15.325ks    2697.e9   32.368%  1.69
```

Fig. 8: Analysis of time behavior of modules and functions

After the long-term trace is started, the TRACE32 software analyzes the time behavior of the functions. The following are analyzed: the runtime and number of clock cycles during the complete test run, the function's share of the total runtime, and the average "clocks per instruction" (see Figure 8).

A detailed analysis of the individual function also shows the time behavior of the program lines (see Figure 9).



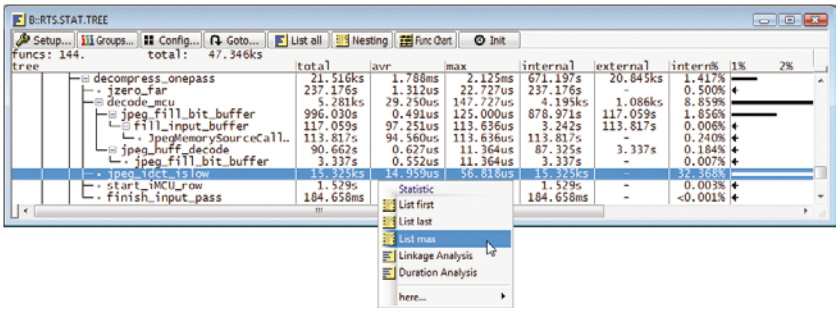
```
[B::Data.List 0xf6a4 /ISTAT SAMPLES]
Step  Over  Next  Return  Up  Go  Break  Mode  Find:            jidctint.c
  samples    time      ratio     addr/line  source

 23405684.  265.973s    0.561%        181      if (inptr[DCTSIZE*1] == 0 && inptr[DCTSIZE*2] == 0 &&
 19724200.  224.138s    0.473%        182          inptr[DCTSIZE*3] == 0 && inptr[DCTSIZE*4] == 0 &&
 20978599.  238.392s    0.503%        183          inptr[DCTSIZE*5] == 0 && inptr[DCTSIZE*6] == 0 &&
 16204910.  184.146s    0.388%        184          inptr[DCTSIZE*7] == 0) {
                                                 /* AC terms all zero */
 17911830.  203.543s    0.429%        186        int dcval = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]) <<

                                                 wsptr[DCTSIZE*0] = dcval;
  4471747.   50.815s    0.107%        189        wsptr[DCTSIZE*1] = dcval;
  4471747.   50.815s    0.107%        190        wsptr[DCTSIZE*2] = dcval;
  4471747.   50.815s    0.107%        191        wsptr[DCTSIZE*3] = dcval;
  4471747.   50.815s    0.107%        192        wsptr[DCTSIZE*4] = dcval;
  4471747.   50.815s    0.107%        193        wsptr[DCTSIZE*5] = dcval;
  4471747.   50.815s    0.107%        194        wsptr[DCTSIZE*6] = dcval;
```

Fig. 9: Details of time behavior of program lines of a function

If the analysis intermittently exceeds the defined maximum time, the cause must of course be found.

Troubleshooting

The long-term trace can be configured so that the trace packages are saved in a file at program runtime. With a data volume of 5 GBytes/hour, about four days of the program runtime can be recorded on an average hard disk. Using fast, sophisticated search functions, the TRACE32 software can then search the trace recording for the excessive function run and show it in a detailed analysis (see Figure 10).



```
B::RTS.STAT.TREE
Setup...  Groups...  Config...  Goto...  List all  Nesting  Func Chart  Init
funcs: 144.        total:    47.346ks
tree                                  total        avr         max      internal    external   intern%  1%     2%
   ├⊟ decompress_onepass            21.516ks    1.788ms     2.125ms    671.197s    20.845ks    1.417%
   │ ├· jzero_far                   237.176s    1.312us    22.727us    237.176s           -    0.500%
   │ ├⊟ decode_mcu                   5.281ks   29.250us   147.727us     4.195ks    1.086ks    8.859%
   │ │├⊟ jpeg_fill_bit_buffer       996.030s    0.491us   125.000us    878.971s    117.059s    1.856%
   │ ││└⊟ fill_input_buffer         117.059s   97.251us   113.636us      3.242s    113.817s    0.006%
   │ ││  └· JpegMemorySourceCall..   113.817s   94.560us   113.636us    113.817s           -    0.240%
   │ │└⊟ jpeg_huff_decode            90.662s    0.627us    11.364us     87.325s      3.337s    0.184%
   │ │  └· jpeg_fill_bit_buffer       3.337s    0.552us    11.364us      3.337s           -    0.007%
   │ ├· jpeg_idct_islow             15.325ks   14.959us    56.818us    15.325ks           -   32.368%
   │ ├· start_iMCU_row               1.529s                              1.529s           -    0.003%
   │ └· finish_input_pass          184.658ms                            184.658ms          -   <0.001%
```

Fig. 10: If necessary, details on the longest function run can quickly be displayed

New Supported Processors

Andes Technology	**LA-3756 (ANDES)** • N9/N10/N12
ARM	**LA-7843 (Cortex-A)** • Cortex-A9 Single Core • Cortex-A9 MPCore
Broadcom	**LA-7760 (MIPS32)** • BCM3556 • BCM7325 • BCM471X **LA-7761 (MIPS64)** • BCM1280/BCM1480
CEVA	**LA-3711 (CEVA-X)** • CEVA-X1641 **LA-3774 (TeakLite-III)** • CEVA-TL3210
Freescale	**LA-7732 (ColdFire)** • MCF5227x/MCF525x **LA-7735 (DSP56300)** • DSP56720 **LA-7733 (MCS08)** • MC9S08ACx/DVx **LA-7734 (MPC5200)** • MPC5121/MPC5123 **LA-7753 (MPC55xx)** • MPC560xx • MPC5633M • MPC5668 • MPC5674 **LA-7764 (PowerQUICC III)** • QorIQ **LA-7736 (MCS12X)** • S12P • S12XF • S12XS
Infineon	**LA-7759 (C166S V2)** • XC2267M-104F • XC2287M-104F • XC2387M-104F **LA-7756 (TriCore)** • TC1736/TC1736ED • TC1767/TC1767ED • TC1797/TC1797ED

Luminary Micro	**LA-7844 (Cortex-M)** • LM3S3700 Series • LM3S5600 Series • LM3S5700 Series
Marvell	**LA-7742 (ARM9)** • 88F5082 • 88F5180N • 88F6082 • 88F6180 • 80F6281
Microchip	**LA-7760 (MIPS32)** • PIC32
Micronas	**LA-7760 (MIPS32)** • VCTH
MIPS	**LA-7760 (MIPS32)** • MIPS74
NEC	**LA-7765 (ARM11)** • NaviEngine **LA-7835 (V850)** • V850E/V850DX3
NXP	**LA-7742 (ARM9)** • LPC29xx **LA-7844 (Cortex-M)** • LPC17xx
Renesas	**LA-7758 (SH)** • SH7786 • SH7722/SH7723 • SH7763 • SH4A-Multi
STMicro-electronics	**LA-7844 (Cortex-M)** • STM32F102 **LA-7836 (MMDSP)** • Nomadik STn8820 **LA-7753 (MPC55xx)** • SPC560x • SPC563M
Tensilica	**LA-3760 (XTensa)** • Xtensa 7

Serial GigaBit Trace Interface

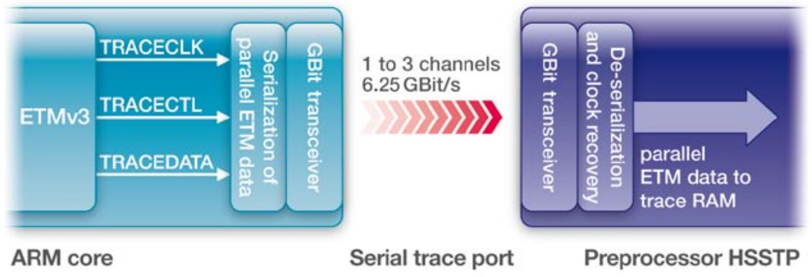

ETMv3
TRACECLK
TRACECTL
TRACEDATA
Serialization of parallel ETM data
GBit transceiver
1 to 3 channels 6.25 GBit/s
GBit transceiver
De-serialization and clock recovery
parallel ETM data to trace RAM
ARM core
Serial trace port
Preprocessor HSSTP

Fig. 11: Block diagram of HSSTP trace for the ARM ETM

Fig. 12: Preprocessor HSSTP

Serial Trace interface resolves two issues at once:

1. Fewer pins are needed for serial transmission.

2. A differential transmission permits higher data rates.

Using only three trace channels, the contents of a complete DVD could be transmitted in less than 3 seconds. This is an impressive example of how fast the serial transmission performance actually is.

Lauterbach is convinced of the benefits of this concept and started work on its technologically ambitious high-speed serial trace project back in 2007. The trace has now been available and in use by customers since the middle of 2008.

Currently, Lauterbach supports the *High Speed Serial Trace Port* – HSSTP for short – from ARM. A development of the *High Speed Trace Port* of QorIQ (e500 Power Architecture) from Freescale is already in the planning stage.

The "Preprocessor HSSTP" (see Figure 12) is designed for a maximum of four high-speed channels. The following transmission rates are supported:

- 6.25 GBit/s per channel with up to 3 channels
- 3.125 GBit/s per channel with 4 channels

The trace data is provided via a custom connector system from Samtec (ERF8, 40 pins).

For transmission, ARM-HSSTP uses the *Xilinx Aurora Protocol*. The parallel trace data is 8b/10b coded and serialized on the ARM core. Differential GBit transceivers send the data flow by cable to the "Preprocessor HSSTP" from Lauterbach, which recovers the original parallel trace data from the serial transmission (see Figure 11).

The large volume of trace data obviously requires a correspondingly large trace memory. This is available from the PowerTrace II with a memory extension of up to 4 GBytes.

Parallel Trace Interfaces

In 2008, support for the parallel trace interfaces was expanded to several new processor families. The table below shows a summary.

Preprocessor AutoFocus II for Parallel Trace Interfaces
Preprocessor AutoFocus II for ARM ETM
Preprocessor AutoFocus II for CEVA-X
Preprocessor AutoFocus II for MicroBlaze
Preprocessor AutoFocus II for PPC4xx
Preprocessor AutoFocus II for SHx
Preprocessor AutoFocus II for StarCore
Preprocessor AutoFocus II for TeakLite-III
Preprocessor AutoFocus II for TMS320C55x
Preprocessor AutoFocus II for TMS320C64x+

Debugging with ARM CoreSight

ARM CoreSight is a good example of the debug and trace concepts for heterogeneous multicore processors.

To process the many tasks within an embedded system, processors that contain different core types are increasingly used. To be able to properly debug such a system, two conditions must be met:

1. The multicore processor must have suitable on-chip debug and trace logic.

2. The development environment must support debugging of the individual cores and also the overall system with intelligent test and analysis functions.

This article describes how the TRACE32 development environment meets these requirements in conjunction with the CoreSight on-chip debug and trace technology.

What Is CoreSight?

CoreSight is the name of the on-chip debug and trace technology provided specially by ARM for multicore processors. However, CoreSight is not designed as a fixed logic block but rather, like a construction kit it provides many different components. In this way, the designer of the multicore processor can define the scope of the functions provided for debugging and tracing. CoreSight offers great freedom of configuration. Integrating suitable debug and trace options on the processor often requires the specialist knowledge of the tool manufacturer. Our experts have been advising developers worldwide on this subject for many years during the design phases of the latest generations of processors.

The construction kit concept of CoreSight naturally has an effect on the development tool used. If the tool knows the processor and its CoreSight component configuration, debugging is very simple. For new processors the construction kit concept requires the tool to have a high degree of flexibility. Although CoreSight configuration information can be read from the processor, it is still often necessary to clarify details of the implementation with the processor's designer.

A heterogeneous multicore processor consisting of the RISC cores ARM11 and Cortex-A as well as a Ceva-X DSP was chosen for the following examples.

CoreSight Debug

For processors with CoreSight, all cores are debugged over a joint JTAG interface. A development environment for our specimen processor consists of the following TRACE32 products (see Figure 13):

- A universal PowerDebug module connected to the host over a USB or an Ethernet interface

- A debug cable with licenses for the ARM11, Cortex-A, and Ceva-X architectures

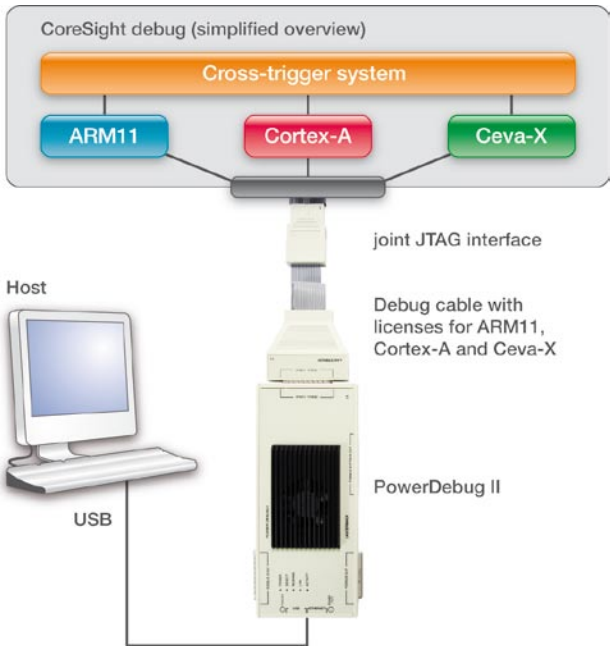

CoreSight debug (simplified overview)
Cross-trigger system
ARM11
Cortex-A
Ceva-X
joint JTAG interface
Host
Debug cable with licenses for ARM11, Cortex-A and Ceva-X
PowerDebug II
USB

Fig. 13: A TRACE32 development environment for CoreSight Debug

In heterogeneous multicore processors, the individual cores usually work on their tasks relatively independently of each other. It therefore makes sense to start a separate TRACE32 instance to debug each core (see Figure 14 on the next page). »

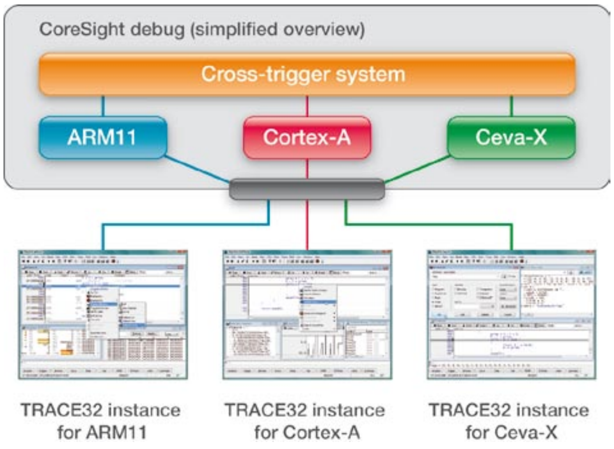

CoreSight debug (simplified overview)
Cross-trigger system
ARM11
Cortex-A
Ceva-X
TRACE32 instance for ARM11
TRACE32 instance for Cortex-A
TRACE32 instance for Ceva-X

Fig. 14: A separate TRACE32 instance is started to debug each core

However, to test that the cores are working properly together, it must be possible to run debugging across the cores. For this purpose, CoreSight provides a cross-trigger system that enables synchronous debugging of all cores: If a core stops at a breakpoint, the other cores are also stopped synchronously. This means the user can easily visualize the context of the individual cores at any selected place in the program.

In addition to this basic function for multicore debugging, TRACE32 can provide other useful debug functions depending on the CoreSight configuration. See the box on the right for a summary of all TRACE32 features for CoreSight Debug.

CoreSight Trace

A common interface is also provided for the trace information from all cores. Under CoreSight, a component for generating trace information can be assigned to each core. For our specimen processor, these are the following components:

- ARM ETM for the ARM11 and the Cortex-A
- Ceva-X ETM for the Ceva-X (see also Figure 15)

Every trace component generates information about the instructions its core has executed and the data accesses that have been made. To provide this trace information at the joint interface, the Funnel combines the trace data into a single data stream. This is then output at a trace port or stored in an on-chip trace memory.

TRACE32-Features for CoreSight-Debug

- Flexible support for multicore processors with CoreSight; Lauterbach offers debuggers for all ARM/Cortex cores as well as a wide range of DSPs
- Debugging over the JTAG interface and the Serial Wire Debug Port
- Runtime access to the physical memory and the peripherals register
- Synchronous debugging of all cores and the peripherals
- The power-down mode of a core has no effect on the debugging of the other cores

Off-Chip Trace Port

Using 18 processor pins (16 pins for the actual trace information and two for control signals), the trace data of all cores can be output to an external trace tool. For

»

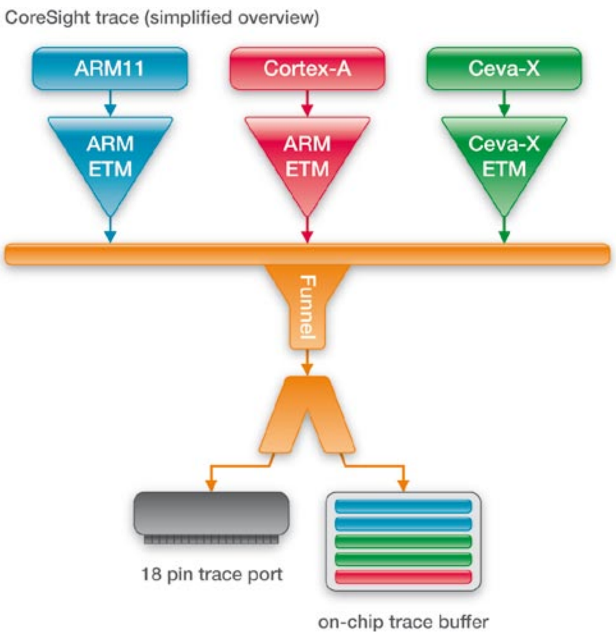
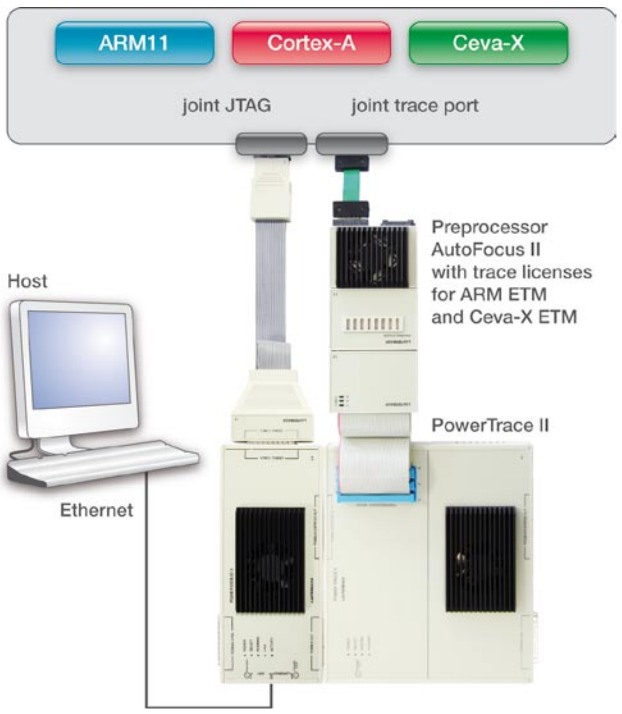


Fig. 15: Every core generates its own trace information



ARM11
Cortex-A
Ceva-X
joint JTAG
joint trace port
Host
Preprocessor AutoFocus II with trace licenses for ARM ETM and Ceva-X ETM
PowerTrace II
Ethernet

Fig. 16: A TRACE32 development environment for CoreSight Debug and CoreSight Trace

off-chip recording and analysis by TRACE32, the following products must be added to the development environment in Figure 13 (see Figure 16):

- A universal "PowerTrace II" module that provides up to 4 GBytes of trace memory
- An "Preprocessor AutoFocus II" for accessing the trace data at the trace port. In this case, the Preprocessor AutoFocus II must contain trace licenses for the ARM ETM and the Ceva-X ETM.

On-Chip Trace Memory ETB

A pin-saving alternative to the trace port is the on-chip trace memory known as the CoreSight *Embedded Trace Buffer* (ETB). However, its capacity is much smaller than an external trace tool – normally only 2 to 8 KB.

If the trace data is saved in the ETB and then read over the JTAG interface, the debug cable in Figure 13 must also contain trace licenses for the ARM ETB and the Ceva-X ETB.

Trace Analysis

After recording, the developer can display and analyze the trace data for each individual core. For this purpose, each TRACE32 instance reads its trace data from the common trace memory.

To analyze the interaction of the cores, their trace displays can be configured to show the trace entries of all cores in a direct time relationship. For example, if a trace entry is selected in the ARM11 instance, the other two TRACE32 instances mark the instruction that was executed by their core at that moment.

Similar to the debug options, the trace options available in TRACE32 depend on the current CoreSight configuration. The trace options ease systematic trouble shooting and allow an overall analysis of system performance. For a summary of the trace features, see the box below.

TRACE32-Features for CoreSight-Trace

- Flexible support for multicore processors with CoreSight; TRACE32 supports the analysis of trace information for the ARM ETM and a wide range of DSP ETMs
- Trace of bus cycles of the AMBA AHB bus
- Trace of data output of the application with the help of the *Instrumentation Trace Macrocell* (ITM)
- Output of trace data at a trace port or storage in the on-chip trace memory
- The trace data generation components can activate each other by means of the cross-trigger system
- Time-correlated visualization of trace data for the individual cores
- Code coverage and comprehensive runtime analyses

Logic Analyzer with 256 MegaRecords

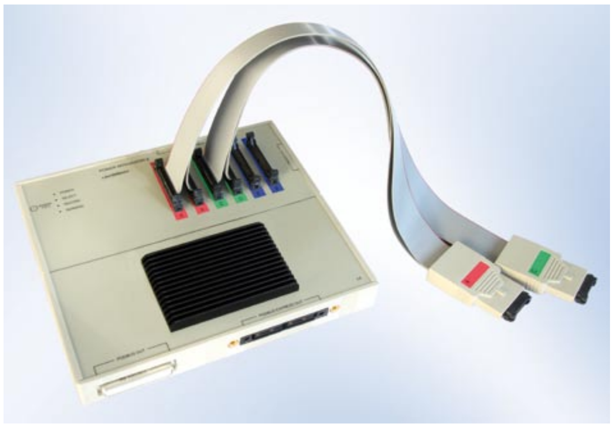


Fig. 17: PowerIntegrator II logic analyzer

PowerIntegrator II, which will be presented at ESC 2009, is the answer to our customers' frequent requests for longer recording times on the Lauterbach logic analyzers. Equipped with up to 4 GBytes of memory, it provides 256 MegaRecords for each of 102 channels. An option for recording 1 GigaRecords with a reduced number of channels is already planned.

The logic analyzer product line of Lauterbach now consists of three devices: PowerProbe, PowerIntegrator, and the new PowerIntegrator II. All devices are equipped for recording control with a trigger unit. Three counters and four trigger levels permit the definition of complex trigger conditions. Of course it supports cross-triggering with a Lauterbach debugger or a real-time trace tool. This makes it easy to stop the complete development environment with everything synchronized, irrespective of which device detected the defined trigger condition.

All logic analyzers offer the following options:

- Recorded raw data can be formatted for a protocol analysis.
- Several channels can be combined to form a memory bus. With the symbol information from the debugger, memory addresses can be displayed using symbols for easier interpretation.
- The recording of current and voltage permits an analysis of the energy consumption of the application.

A typical Lauterbach development environment with logic analyzer then consists of: A debugger, a real-time trace tool for recording the program/data flow and a logic analyzer for recording application-relevant digital/analog signals.

The recordings of the real-time trace and the logic analyzer can be time-correlated. This means that all details of the application can be checked at a glance.

The new PowerIntegrator II is designed to be deployed where long recording times are required. A typical application is the analysis of serial protocols. Due to the large data volumes that have to be transmitted to the host for the PowerIntegrator II, the GBit Ethernet interface of the PowerDebug II should be used.

	PowerProbe	PowerIntegrator	PowerIntegrator II
Trace depth	256 K records	512 K records	256 000 K records
No. of channels	64/32/16	204/102	102/51
Timing mode sampling	100/200/ 400 MHz	250/500 MHz	250/500 MHz
State mode sampling	100 MHz	200/400 MHz	200/400 MHz
Adaptation	Clip set	Mictor, Samtec, standard header, clip set	Mictor, Samtec, standard header, clip set
Extras	Stimuli Generator, FPGA trace	—	Stimuli Generator

The Latest on RTOS Debugging

The TRACE32 debug environment includes a configurable RTOS debugger to provide the user with symbolic debugging capability in real-time operating systems. Adaptations for all popular operating systems are already included, at no extra cost, within the standard Lauterbach debug environment.

There's a lot of news this year about operating systems. The most important innovation in 2008 was certainly the increasing use of SMP operating systems for embedded designs. For a list of supported multicore processors, see the table bottom left.

Every year presents the arrival of a whole new range of operating systems on the market; traditional operating systems receive an update – for more on this subject, see the table right.

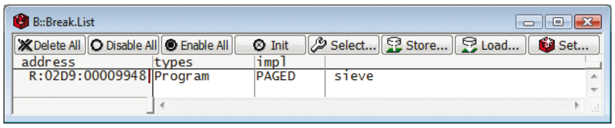
Linux: Run & Stop Mode Debugging

- Switching between JTAG and GDB debugging on the TRACE32 GUI is now also supported for the SH architecture.
- For the ARM architecture, the *debug communications channel* – DCC for short – can now also be used simultaneously for GDB debugging and the Linux terminal window.

Supported SMP Operating Systems	
Linux, QNX, Symbian, ThreadX	
For the Following Processors	
ARM11 MPCore	ARM
Cortex-A9 MPCore	ARM
MIPS34K	MIPS Technologies
MPC8572	Freescale
MPC8641D	Freescale
SH7786	Renesas

Linux: Paged Breakpoints

The TRACE32 software and a suitable Linux patch enable a software breakpoint to be set for program code that has not yet been loaded (paged breakpoint). Paged breakpoints are currently possible for the ARM and MIPS architecture.



```
address           types        impl
  R:02D9:00009948 Program      PAGED       sieve
```

Fig. 18: Paged breakpoint

NetBSD: Library Support

An extension of the TRACE32 RTOS debugger for the NetBSD operating system now permits not only the debugging of processes but also the debugging of library functions.

New Supported RTOS	
ARTX-166 für C166	available
Linux for Andes, ARC and MicroBlaze	available
LynxOS 4.0 for PowerPC	available
LynxOS 5.0 for PowerPC	planned
LynxOS-SE for PowerPC	planned
Nucleus for Andes and MicroBlaze	available
OKL4 for ARM	planned
QNX 6.4 for ARM, PowerPC, SH and XScale	planned
RTEMS for ColdFire	available
RTX-ARM for ARM	available
ThreadX for Xtensa	available
Windows CE 6.0 for MIPS	available
Xikernel for PowerPC	planned
µClinux for Blackfin	available

CombiProbe for STM32F1xx

CombiProbe is the ideal development tool for the STMicroelectronics STM32F1xx processors with ETM and ITM. Debug and trace options which up to now have only been available on the more powerful ARM processors are now offered on the Cortex-M3 at low price and compact size.

The CombiProbe is a combination of a special debug cable and a 128-MB trace memory. It is usually connected to a universal PowerDebug module.

For STM32F1xx processors, the CombiProbe can be used for the following:

- Debugging over standard JTAG and Serial Wire Debug Port
- Tracing the program flow over the ETMv3
- Tracing selected data accesses or application-specific data over the ITM

CombiProbe	
Luminary Micro	Stellaris processors with ITM
Microchip	PIC32 with IFLOW-Trace (program flow)
NXP	LPC17xx with 4-bit ETMv3 (program flow) and ITM
STMicro-electronics	STM32F1xx with ETMv3 (program flow) and ITM

The CombiProbe can be configured so that during recording the trace data is transferred to the host. This data can be processed in real-time and/or stored on disk. The table above lists further processor architectures supported by the CombiProbe.

KEEP US INFORMED

If your address has changed or if you no longer want to be on our mailing list, please send us an e-mail to:

info_us@lauterbach.com

Branches Around the World

Germany
Lauterbach GmbH
info@lauterbach.com
Phone +49 8102 9876 0

USA East
Lauterbach Inc.
info_us@lauterbach.com
Phone +1 508 303 6812

USA West
Lauterbach Inc.
info_us@lauterbach.com
Phone +1 503 524 2222

France
Lauterbach S.A.R.L.
info_fr@lauterbach.fr

UK
Lauterbach Ltd.
info_uk@lauterbach.com
Phone +44 1256 333 690

Italy
Lauterbach Srl
info_it@lauterbach.com
Phone +39 02 45490282

China
Suzhou Lauterbach Technologies Co. Ltd.
info_cn@lauterbach.com
Phone +86 512 6265 8030

Japan
Lauterbach Japan Ltd.
info@lauterbach.co.jp
Phone +81 45 477 4511

And represented by competent partners in all other countries!