TRACE32®
IN-CIRCUIT EMULATORS & DEBUGGERS
ONE SYSTEM FITS ALL!

NEWS

2003

Visit www.lauterbach.com for more information

NEWS FOR THE EMBEDDED SYSTEMS CONFERENCE

An optimistic outlook for 2003

Increased sales

2002 was a highly successful year for Lauterbach with sales up by 20%. Whenever we spoke to customers or colleagues in the industry the reaction was one of surprise. Sales increases in 2002? How could that have been possible in such a difficult business environment? 2002 was not such a good year for the majority of companies in the high-tech sector and the market for development tools was unlikely to have grown last year. There is, therefore, every indication that Lauterbach gained a greater share of the market as a whole. In other words, in 2002 we again succeeded in consolidating our market position. This was very plain to see in the UK where Lauterbach's new branch posted excellent sales figures and turned a profit in its very first year.

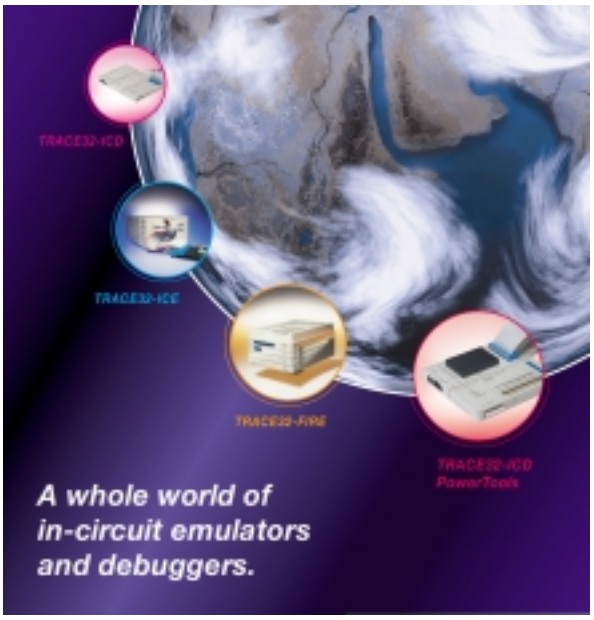
Of course nobody can predict whether 2003 will be another successful year but Lauterbach is extremely well placed as it goes into the new year.

Own branch in Japan

Japan is a large and important market as everyone knows. It is also well known for being a very closed and difficult market for foreign companies. This makes it all the more important for a company to be represented with its own branch in Japan if it wants to market its products there successfully. Lauterbach has managed to attract Japanese employees with great experience in the embedded market for the branch operation that we will be establishing in Japan in 2003. This is an important step for further strengthening the company's global presence.

Development team strengthened

At the end of 2002 and the beginning of 2003 we recruited a number of new employees for our headquarters in Hofolding. They will strengthen the development department especially in the field of VHDL designs in TRACE32 products and the further development of JTAG tools. We are therefore confident that all projects planned for 2003 can be implemented speedily and on schedule.



New products

As you will see on the following pages there are many new products and ideas at Lauterbach. Without doubt the key focal points are multicore debugging and an extended range of debuggers for DSPs.

At the Embedded Systems Conference in San Francisco Lauterbach will be introducing a brand new debugger for the OMAP and TMS320C55x DSPs from Texas Instruments. We would like to invite you to visit us on our exhibition stand for a relaxed and informative meeting.

LAUTERBACH

Multicore debugging

The debugging of system-on-chip designs containing several cores places new demands on development tools. Ideally chip designers and tool producers should agree quickly on suitable techniques that guarantee the user rapid availability of high-quality development tools with free selection of the core combination.

Application-specific ASICs are now being used to an increasing extent within embedded designs. In order to achieve optimum functionality and performance it is becoming more and more common for several cores to be integrated to form a system-on-chip (SoC). The use of a RISC processor combined with a DSP is widespread. Examples are:

- *OMAP processors from Texas Instruments containing ARM925/926 RISC core and TMS320C55x DSP*
- *S-GOLD from Infineon containing ARM926EJ-S and OAK DSP*

The testing and the integration phase of this type of multicore SoC design places new requirements on microprocessor development tools. This article focuses on debugging by addressing several cores via a shared debug interface. We present several approaches and attempt to explain the problems involved.

Addressing several cores via one debug interface

Most modern processors provide on-chip debug logic for debugging. The interface that is most frequently used for controlling the debug logic is defined by the JTAG specification. However, with multicore SoC designs a separate JTAG interface is not provided for every core in order to save on the number of pins and so cut costs. Instead, a joint JTAG interface has to suffice for debugging all cores. This raises the question of how the debugger can address several cores via the joint JTAG interface and how synchronous debugging can be implemented.

1. One debugger supporting all cores on the SoC

At first glance this would appear to be the ideal solution for the developer. The main advantages of this solution are as follows: one development tool for all cores and hence you only need familiarization with one user interface and product philosophy. This is certainly the best approach to adopt for SoCs that are produced in high volumes and are used for a large number of developments. An example of this type of solution is the OMAP debugger from Lauterbach that allows debugging of the ARM925/926 core as well as the TMS320C55x DSP.

In contrast there is also a market trend towards individualized SoCs that are used for the development of only one product. A new SoC that achieves even more optimised performance and functionality is then created for the next product. If the developer also wants full freedom in the selection of the integrated cores, he will also need the services of a tool producer who can provide an optimum product for every architecture that is employed. Naturally the debug tool should also be available fully tested by the time the first chip is produced from the wafer.

Although this solution is obviously the best solution available, the chances of it being implemented cheaply and more especially quickly while maintaining a high standard of quality are rather unlikely.

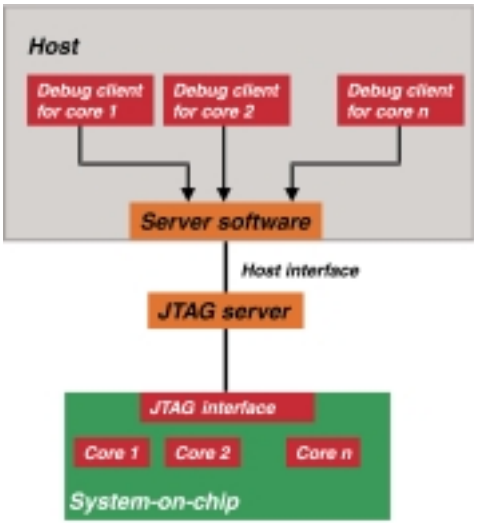


Figure 1: JTAG server solution

2. Using a JTAG server

Highly developed debuggers are already available today for the majority of cores. We might therefore ask, why not connect the existing debuggers to a JTAG server that takes care of the addressing of the individual cores via the joint JTAG interface?

One can imagine a solution of this kind being implemented as follows: Interface hardware is connected in front of the joint JTAG interface. This hardware is controlled by server software on the host. The individual debug client now sends its communication request to the server by means of a remote procedure call. This guarantees exclusive access to the joint JTAG interface and forwards the communication request to the individual core. Figure 1 shows the basic principle of a server solution of this type.

In this case the debugger uses a software interface to the JTAG server instead of dedicated JTAG hardware.

This approach, although it appears very elegant at first sight, similarly does not offer a complete solution to the problem. A number of semiconductor manufacturers already offer server solutions for core groupings in their product range, but there is no server standard in sight that might offer a vendor-neutral solution.

The JTAG server solution also proves to be rather inflexible when individual semiconductor manufacturers have expanded the JTAG signals of their core in order to offer additional debug functionality. In a multicore development environment, for example, it would be desirable for each core to have a stop request signal making it possible to stop the core immediately. Another requirement would be to indicate that a core has stopped by means of a stop indication signal. Ideally then both signals could be used for synchronous stopping of all cores in a multicore application.

With free selection of an optimum core combination a number of additional signals will quickly appear. The easy option would be to simply let these signals go by the board and dispense with the additional functionality. However, these signals are often necessary or at least very helpful and so need to be included. The JTAG server would therefore still have to be matched to the specific application and tested.

Another critical point, especially for real-time applications is the large volume of data that has to be handled by the JTAG server. This slows the reaction time of the individual core accordingly. Therefore, in order to meet the system's real time requirements it is necessary to pack time-critical functions into the hardware, which then brings us back to the application-specific server.

An optimum server solution is therefore one in which all the cores used come from the same semiconductor manufacturer, this manufacturer then supplies the JTAG server and at the same time the matching debuggers. Application-specific servers on the other hand require agreements between the producers of the debuggers. This is something that usually proves difficult because of the competitive situation and is also time-consuming and cost-intensive. An added factor is that it will be necessary to adapt the server to cater for every new core combination.

3. Completely independent debuggers at the joint JTAG interface

This approach represents a simple and open solution in which the best possible core combination can be freely selected for the particular application. During development and integration the developer can resort to fully optimised debuggers without the need to match the development environment to a specific application.

The basic idea in this case is that an independent debugger is con-

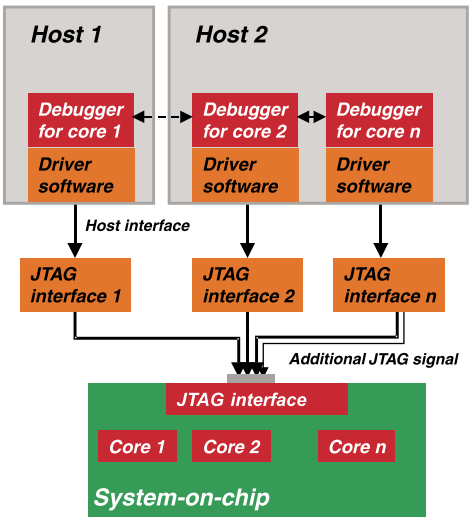


Figure 2: Completely independent debuggers at the joint JTAG interface

nected to the joint JTAG interface for every core. In order for this approach to function, each debugger must deactivate its JTAG driver when it is not exchanging data with its core. Figure 2 on page 3 illustrates this approach.

Since there are now several independent debuggers using the same JTAG interface it is necessary to ensure that only one debugger operates at the interface at any one time. This can be automated through a system whereby the debug tasks on the host use a semaphore system to define who is given exclusive access to the JTAG interface. Another possible alternative would be to use hardware semaphores between the debuggers.

The question of synchronous starting and stopping of all cores can be solved simply and effectively via special logic on the SoC. For synchronous stopping, for example, every core can be equipped with two additional signals:

- *A stop request signal that enables the core to be stopped immediately*
- *A stop indication signal that indicates that the core has stopped*

These signals could then be combined on the SoC via a matrix. Each core can then set whether it wants to stop or continue running if another core stops, for example by setting a memory-mapped control register. Figure 3 illustrates a matrix of this type. This matrix could also be easily expanded to include the peripherals of the individual cores. The main advantage of this solution is the very high degree of synchronization when stopping and the provision by the chip designer of a standard interface for the debugger manufacturer.

This solution offers a relatively simple procedure for chip designers and tool producers. If a few basic preconditions are provided, any cores can be integrated in the SoC and the associated development environment can be selected freely.

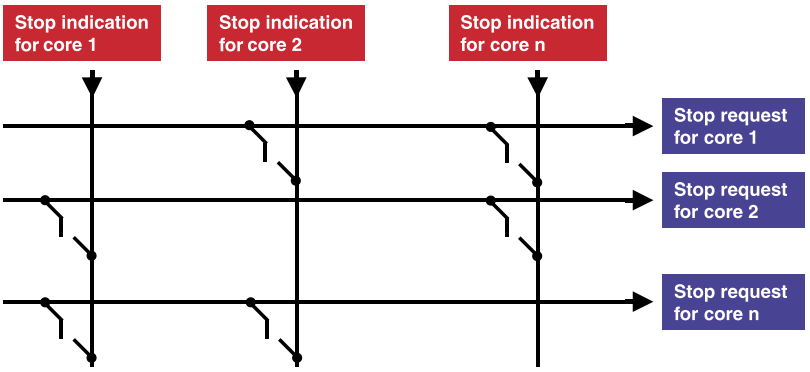


Figure 3: Matrix for synchronous stopping of several cores

The advantages in this case can be summarized as follows:

- *No additional agreements between the tool producers are necessary for this approach. High-performance development tools are available immediately.*
- *The solution that we have presented does not require any application-specific modifications on the part of the tool producer. Standard tools can be employed which can be reused in other projects.*
- *The apparent disadvantage that a complete debug system must be acquired for every core turns out at a second glance to be more of an advantage. Normally during the development phase only one core is being tested at a time and the debuggers can be used individually for this. The only time when all debuggers have to be used together is during the integration phase when testing the interactions between the cores.*
- *Lauterbach already supports a wide array of cores with its debuggers. A standard user interface and a common product philosophy therefore already exist for using these debuggers for multicore SoC designs.*

As long as multicore debugging remains in its infancy and no general standard is defined Lauterbach regards this third solution as the most effective. Debuggers from Lauterbach are designed to meet all the requirements of operating at a joint JTAG interface.

Multicore debugging with Lauterbach tools

1. Required debugger hardware

Several cores can be debugged via a joint JTAG interface simply by plugging several Lauterbach debuggers together and connecting them to the host system via one standard interface (LPT, USB or Ethernet). Each debugger has its own JTAG cable.There are two alternatives for connecting the JTAG cables:

Figure 1: Two independent Lauterbach debuggers that debug an SoC with two cores via a joint JTAG interface

- *The target system provides a separate connector for every JTAG cable.*
- *There is only one JTAG connector on the target system. In this case a suitable adapter must be used that provides connectors for all JTAG cables. This solution is shown in Fig. 1.*

Of course it is also possible to add the matching trace modules for each core.

2. Configuration

There are a number of points a user has to watch out for when configuring their Lauterbach debugger.

The key factor for the configuration is how the cores are arranged in the SoC. The following description refers to a simple yet frequently used configuration where the JTAG ports of the individual cores are arranged serially (daisy chained). See also Fig. 2.

The first question is how do the individual debuggers address their core?

According to the JTAG convention there is an option to generate sequences for the scan chain that ensure that individual TAP controllers behave neutrally (BYPASS). To address a specific core the debugger must therefore be configured in such a way that when generating the scan chain BYPASS sequences are created for all TAP controllers before and after this core (see IRPOST, IRPRE, DRPOST and DRPRE in Fig. 3).

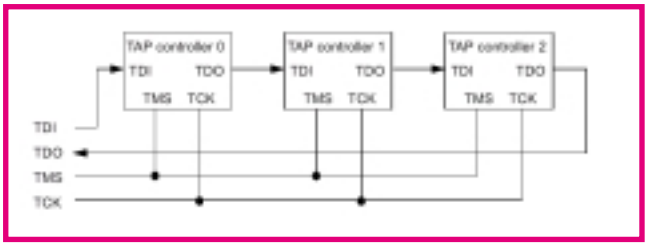


Figure 2: Serial arrangement of the JTAG ports (daisy chained)

The second question is how do the individual debuggers behave if they do not need access to the JTAG port?

Any debugger that is not currently using the joint JTAG interface must switch its lines into the tristate mode. A suitable state is then selected for the TAP controller (TAPState) so that the debugger can restart at a defined point the next time the JTAG lines are activated. The debugger sets the TAP controller to this state before activating the tristate mode and will then expect the TAP controller to be in this state when the JTAG lines are reactivated.

The most important JTAG lines must be terminated with resistors so that no lines are left open in the tristate mode and the TAP controller is not unintentionally reconfigured. To complete the configuration it is now necessary to define which debugger is the master over the Reset lines of the SoC (Slave).

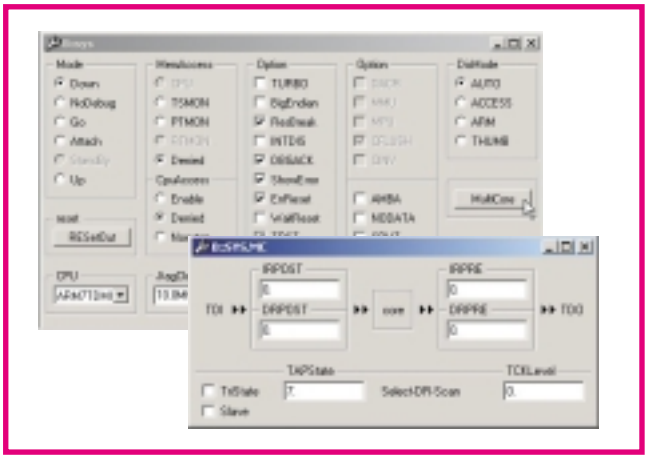
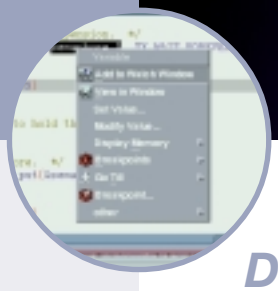


Figure 3: Configuration of a debugger sharing a joint JTAG interface with other debuggers

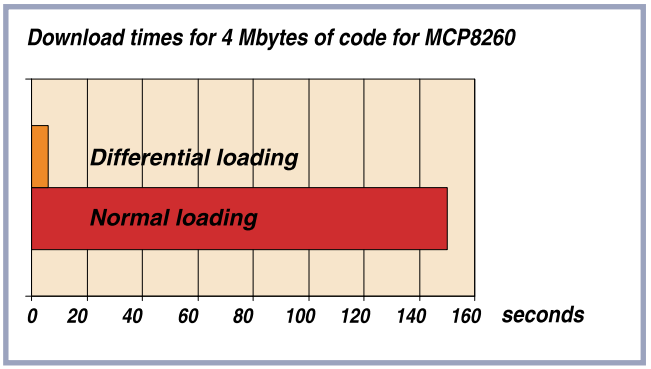


Differential loading

Lauterbach is rolling out a new methodology for loading large programs. This makes it possible to load into the development tool only those parts of the program that have actually changed. Since this normally only involves a fraction of the entire program, download times are reduced by a average factor of 30.

A typical software development cycle consists of debugging, rectification of errors, recompiling, reloading and debugging again. When large programs are tested on a processor where the architecture of the on-chip debug interface does not allow high-speed loading the developer has to spend a lot of time waiting.

In order to minimise waiting times during software loading, Lauterbach now offers the option of only loading the new parts of the program that have actually changed. The principle of differential loading functions is as follows: The developer enables differential loading as an option when entering the Load command. The loading software of the development tool then identifies the parts of the program that have been changed, compresses them and loads the compressed data into a predefined memory area via the on-chip debugging interface. The loading software then starts a target agent that decompresses the data and ensures that it is written into the correct memory areas.



Differential loading is recommended primarily for the following architectures: PPC7xx, MPC82xx, JANUS and ARM with RTCK

You can find more information about differential loading at:

http://www.lauterbach.com/ lightspeed.html

Software license key

In order to facilitate the administration of software license keys the option now exists for all TRACE32-ICDs from December 2002 onwards to store the license key directly in the debug cable.

The individual debug cable can now be used without any problem on a notebook or in a test environment where access to a license file is not possible.

Real-time kernels now supported

- ***QNX from QNX Systems Ltd.***
  for ARM, PowerPC and XScale
- ***µCLINUX***
  for 68k, ARM and ColdFire
- ***eCOS from Red Hat***
  for PowerPC
- ***Remote Debugging for OSE from OSE Systems***
  for ARM
- ***LINUX***
  for ARM and XScale

THE PRODUCT LINE

TRACE32-ICD

New debuggers

In 2003 Lauterbach will also be increasing the number of processors supported by the in-circuit debugger TRACE32-ICD. Apart from the latest ARM cores and the new PowerPC derivatives the main focus this year will be on DSPs which are used primarily in multicore designs.

ROM monitor for ARM

With effect from November 2002 Lauterbach also offers a ROM Monitor for all ARM cores and for the XScale from Intel.

The ROM monitor can be operated either together with a FLASH simulator or via the serial interface. This means that there is now also a TRACE32-ICD debugger for all the ARM cores without a JTAG interface, for example for the ARM8 and the StrongARM.

Licensing of serial ROM monitors

A USB dongle has also been available from November 2002 for the licensing of serial ROM monitors. This provides a solution that allows more flexibility compared with the current system of licensing via the Ethernet address of the host.

ARM	ARM9EJ ARM10 Core ARM11 Core	now Q1/2003 Q3/2003
Avalent	Janus 2	now
DSP Group	TeakLite/OAK	Q2/2003
Hitachi	H8S2319, H8S2329, H8S2339	now
Infineon	S-GOLD (ARM926EJ-S and OAK)	now
Motorola	MPC5500 MGT5100 MGT5200 MPC750FX/CX/CXE MPC74xx MPC8265/MPC8264 MPC8540/MPC8560 MPC827x MPC828x MCS12C32 including on-chip trace	Q3/2003 now Q2/2003 Q1/2003 Q1/2003 Q1/2003 Q2/2003 Q2/2003 Q2/2003 now
STMicro-electronics	Super10	now
Texas Instruments	OMAP (ARM925/926 and TMS320C55x)	now

Software breakpoints in the FLASH

With the aim of making debugging easier and more convenient for the TRACE32-ICD and PowerTools tool families Lauterbach has expanded the FLASH programming so that it is now possible to insert software breakpoints into the FLASH memory and patch the code in the FLASH.

Most processors that permit debugging via an on-chip debugging interface only have a limited number of on-chip breakpoints. Developers who want to test their software in the FLASH memory therefore come up against this limit very quickly.

For this reason, since mid-2002 Lauterbach has also offered the option of inserting software breakpoints into the FLASH memory. In order to minimize the time spent on reprogramming, the TRACE32-PowerView software now also supports the full sectoring of all FLASH types. Also an attempt has been made to reduce the frequency of FLASH reprogramming through intelligent algorithms.

Thanks to the full sectoring of all FLASH types it is now also possible to implement a patch in the FLASH code in a very short time.

Both features are now available for all internal and external FLASH memories and can also be easily used for target-based FLASH programming.

PowerIntegrator

At the "embedded world 2003" (in Nuremberg/Germany, February) Lauterbach will be introducing the PowerIntegrator, a revolutionary concept in logic analyzers that can be easily integrated into the TRACE32 development environment providing users with a means of monitoring all bus activities and port lines on their target system during debugging.

In addition Lauterbach will offer a range of support packages that can get PowerIntegrator up and running quickly:

- ***Board support packages*** *to configure the PowerIntegrator for use on standard evaluation boards.*
- ***Bus support packages*** *to configure the PowerIntegrator for special bus protocols such as SDRAM busses, double data rate busses, etc.*

The PowerIntegrator has 204 freely configurable data channels, 12 of which can also be used optionally as clock inputs. The following sampling rates are possible using active probes:

- ***500 MHz in the timing mode***
- ***200 MHz in the state mode***

Two alternative probes are available for sampling the signals:

- ***Mictor probe*** *with 32 data channels and 2 clock/data channels*
- ***Standard probe*** *with 16 data channels and 1 clock/data channel*

The sampling threshold for the signals can be selected within a range from 0 to 4.0 V.

The PowerIntegrator has a trace memory depth of 256 KFrames. Every entry in the trace memory is provided with a time stamp with a width of 48 bits operating at a resolution of 8 ns.

The following example describes briefly how the PowerIntegrator operates and for illustration purposes we will use the tracing of the program and data flow via an SDRAM bus. The following steps are necessary for this:

PowerIntegrator on the target system

1. Defining which signals form the address and data bus and the configuration of the control signals

The PowerIntegrator offers a universal NAME command for definition purposes that allows individual signals to be grouped together and to be assigned logic names.

2. Prefiltering of the cycles on the SDRAM bus

It is advisable to take only those signals from the activities on the SDRAM bus that can be used for the subsequent generation of the program and data flow and transfer them into the trace memory. This enables the maximum utilisation of the PowerIntegrator trace depth. Setting the transient detection to suitable levels for the control signals on the SDRAM bus will achieve this.

3. Generating the program and data flow with the aid of a suitable script

The program and data flow must now be generated from the prefiltered information recorded in the trace memory. For this purpose it is necessary to use a script language to define which control signals must be activated for a fetch, read or write cycle.

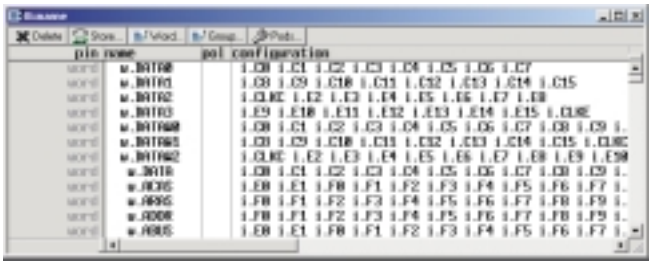
The screenshots on page 9 show this procedure step by step.

Mictor probe

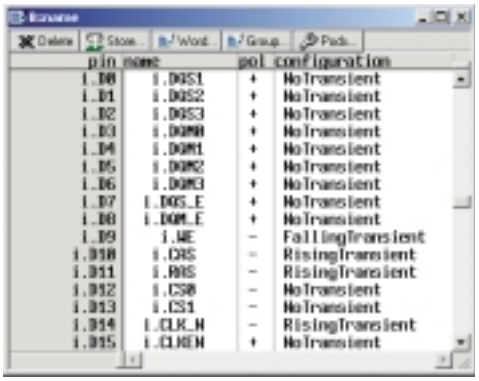
TRACE32-POWERTOOLS

If the PowerIntegrator is operating with a debugger the data information from the fetch cycles can also be disassembled. Then it becomes possible to display the associated high-level language information.

The preparation of the script for generating the program and data flow from the trace data is usually time-consuming, requires a good knowledge of the bus protocol and can therefore be expensive.

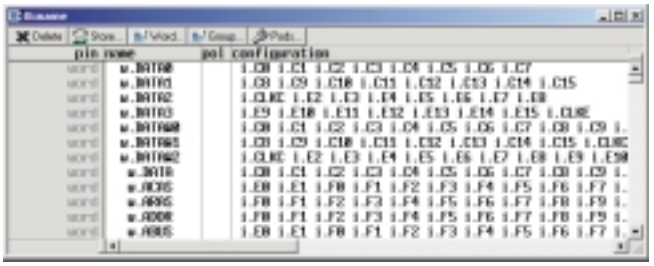


Definition of the signals for the address and data bus, and the control signals



pin name		pol	configuration
i.D0	i.DQS1	+	NoTransient
i.D1	i.DQS2	+	NoTransient
i.D2	i.DQS3	+	NoTransient
i.D3	i.DQM0	+	NoTransient
i.D4	i.DQM1	+	NoTransient
i.D5	i.DQM2	+	NoTransient
i.D6	i.DQM3	+	NoTransient
i.D7	i.DQS_E	+	NoTransient
i.D8	i.DQM_E	+	NoTransient
i.D9	i.WE	-	FallingTransient
i.D10	i.CAS	-	RisingTransient
i.D11	i.RAS	-	RisingTransient
i.D12	i.CS0	-	NoTransient
i.D13	i.CS1	-	NoTransient
i.D14	i.CLK_N	-	RisingTransient
i.D15	i.CLKEN	+	NoTransient

Prefiltering of the cycles on the SDRAM bus through suitable selection of the transient detection



Program and data flow generated from the tracing of the activities on the SDRAM bus

Compared to the above software based solution Lauterbach can offer an alternative:

Through the use of an SDRAM bus support package it is not necessary for the user to actually carry out the three steps previously described. Instead the entire intelligence for generating the program and data flow is loaded into an FPGA. The logic in the FPGA then physically generates the address and data bus and the control signals from the activities on the SDRAM bus.

The use of a support package offers the user the following advantages:

- *The PowerIntegrator is fully preconfigured for the required tracing.*
- *Filtering of the bus cycles is optimized for the bus protocol.*
- *As the address, data bus and the control signals are now physically generated it becomes possible to offer complex real-time triggering. The triggering permits cycles to be selectively filtered out of the program and data flow or can allow the program to be stopped under a defined trigger condition.*

Lauterbach's objective with the roll-out of the PowerIntegrator is to offer a universal trace solution to the market.

*The **software-based approach** provides a highly flexible and open solution which by using a script enables the processing of all port lines as well as the bus activities.*

*With the **FPGA-based approach** and the support packages the emphasis is more on the fast availability of efficient processing of bus and port information.*

Board support packages for standard evaluation boards for ARM, MIPS and XScale architectures are planned for the first half of 2003. A version for the PowerQUICC III will be implemented as soon as the first evaluation board is released to the market.

The first bus support package will be for the SDRAM bus, the other bus architectures will follow. Future availability of support packages will be based upon feedback from customers.

TRACE32-ICD
PowerTools
the complete solution
for system on-chip designs

Visit www.lauterbach.com for more information

Hardware-based code coverage

What advantages does hardware-based code coverage bring compared with the trace-based code coverage possible in the past?

Trace-based code coverage is restricted by the size of the trace buffer and can therefore only be implemented for a short monitoring period. In comparison hardware-based code coverage can also be used in long-duration tests over large program areas.

What is hardware-based code coverage?

Since October 2002 it has been possible to implement hardware-based code coverage with the PowerTrace. A separate memory is provided in the PowerTrace for this functionality from which 2 bits per instruction can be mapped. These two bits can be used to mark:

- *if an instruction has been executed by the program.*
- *if or not a branch has been taken.*

	executed	branch taken
never	0	0
taken	0	1
not taken	1	0
ok	1	1

Marking of branch commands for code coverage

Which processor types are supported for hardware-based code coverage?

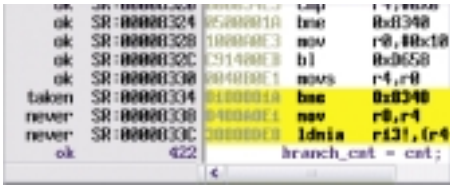
Hardware-based code coverage is currently available for all processors with ARM-ETM (ARM7/9-based cores), also for the SH4 from Hitachi and the ST40 from ST Microelectronics. Expansion to other architectures is constantly being evaluated.

As a basic requirement it must be possible to configure the processor in so that the trace information provided from the program flow at run time contains the branch destinations of both the direct and indirect branches. Since the branch destination information is usually coded and is provided after a time delay, the trace information has to be additionally processed by an FPGA to set program run time markers on the individual instructions.

What is the maximum number of instructions that may be monitored?

The PowerTrace provides storage space for 4 x 2 million instructions. This memory must be accurately mapped. The maximum physical address range for which hardware-based code coverage can be implemented depends essentially on the minimum width of an individual instruction.

With an instruction width of 32 bits, for example, 4 x 8 Mbytes can be analyzed.



USB 2.0

As from March 2003 the PowerDebug Module/Ethernet will be shipped with a USB 2.0 interface.

With a transfer rate of 480 Mbps the USB 2.0 transfers data as fast as using a 100-MBit Ethernet interface.

New feature for PowerProbe

The PowerProbe can now be configured to record a program and data flow trace.

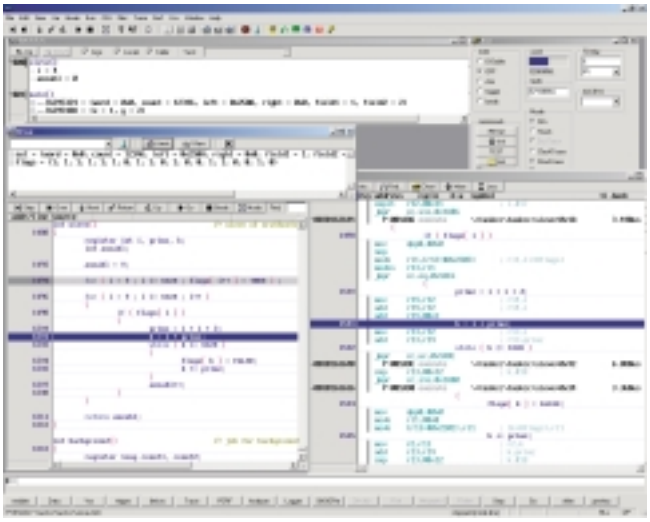
The following steps are required:

1.) Defining which signals form the address and data bus, and configuration of the control signals.

2.) Generation of the program and data flow with the help of a script that defines which control signals have to be active for a fetch, read and write cycle.

TRACE32-POWERTOOLS

NEXUS for Super10



The JTAG debugger for the Super10 from ST Microelectronics has been available since October 2002. We are now able to announce that the development program for the NEXUS debugger has also been completed.

JTAG-Super10 *offers user-friendly assembler and high level language debugging for all standard compilers. Full support is provided for the on-chip break and trigger options.*

NEXUS-Super10 *also provides tracing of the program and data flow in addition to the debugging functions. NEXUS-Super10 has been implemented on the high-performance PowerTrace system which fully supports the functions for run time measurement, performance analysis and code coverage in addition to comprehensive trace analysis. The first processor from the Super10 family to be supported is the ST10R303.*

New casing for TRACE32

Higher frequencies and the demand for modern trace systems requires that we have a continuous development program for our products. For this reason, from March 2003, the following TRACE32 products will be supplied with a new design of packaging.

Standard format dongle with plug-in debug cable

Debug cable

By providing a larger format dongle more space has been created for the increasingly complex logic that has become necessary to control the on-chip debug interface.

With this new packaging the debug cable is no longer permanently soldered to the dongle but is simply plugged in. This allows more flexibility in terms of the cable length, provided that due care is taken over the choice of frequency for the on-chip debugging interface.

Preprocessors and Nexus adapters

Preprocessors and Nexus adapters are now installed in a casing for better protection against contamination, static and prevention of possible short-circuits with the target system.

Preprocessor in casing with Flex-Extension for easier connection to the target system

TRACE32-FIRE
the superfast fully integrated emulator

Visit www.lauterbach.com for more information

TRACE32-FIRE: Latest news

The number of 16-Bit FLASH microcontrollers supported by TRACE32-FIRE will continue to be expanded during 2003.

512K Frame trace memory

With effect from March 2003 the TRACE32-FIRE will also be available with a 512K Frame deep trace memory.

H8S from Hitachi

TRACE32-FIRE will also be extended to support further members of the H8S family in 2003. The H8S/2612 and the H8S/2678 are scheduled for spring 2003.

H8 from Hitachi

From September 2002 the TRACE32-FIRE RISC emulator has supported emulation of the Hitachi H8 family. The following processors are currently supported:

- *H8/3006/3007/3008*
- *H8/3044/3045/3046/3047/3048*
- *H8/3052*
- *H8/3060/3061/3062/3064/3065/3066/3067/3068*

XC16x from Infineon

In 2002 support was announced for the new derivatives of the Infineon XC16x family. The following processors can now be emulated:

- *XC161CJ*
- *XC164CS*

During 2003 Lauterbach is also planning to adapt the TRACE32-ICD in-circuit debugger and TRACE32-FIRE RISC emulator to include all the new family members.

MCS12 from Motorola

Support for members of the 68HC12 and MCS12 family was completed during 2002. New modules were added for the 68HC912DT128 and MCS12DB128.

ST10F276 from ST Microelecronics

With immediate effect the TRACE32-FIRE RISC emulator for the ST10 family also supports the ST10F276.

The ST10F276 has a larger on-chip FLASH memory than the ST10F269 which makes it an attractive solution for many developments. Customers who are already using a TRACE32-FIRE for the ST10F269 can have this emulator converted by Lauterbach to cover the ST10F276.

Please send me information material on:

We use the following processors:

_____

I am interested in the following development tools:

- ☐ ***TRACE32-ICD***
- ☐ ***TRACE32-PowerTools***
- ☐ ***TRACE32-PowerIntegrator***
- ☐ ***TRACE32-FIRE***

Sender:

Name _____

Company _____

Address _____

Phone _____

email _____

LAUTERBACH

Phone: (508) 303 6812 FAX: (508) 303 6813