The future of In-Circuit-Emulation – Code Coverage after “embedded-trace“

Prof. Dr. Christian Hochberger,
Dipl.-Ing. Alexander Weiss,
March 2007

Document No.: FA-EN-070312

Contact Editors:

Prof. Dr.-Ing. Christian Hochberger
Dresden University of Technology
Department of Computer Science
Institute for Computer Engineering
Nöthnitzer Straße 46
01187 Dresden
Germany
Phone: +49 351 463 39625
Fax : +49 351 463 38245
Email: christian.hochberger@inf.tu-dresden.de

Dipl.-Ing. Alexander Weiss
Accemic GmbH & Co. KG
Hochriesstr. 2
81326 Flintsbach
Germany
Phone: +49 8034 90993-12
Fax : +49 8034 90993-27
Email: aweiss@accemic.com

The future of In-Circuit-Emulation – Code Coverage after "embedded-trace"

Ever more complex microcontrollers that are clocked ever faster, open up to new and exciting fields of application, amongst others to security-sensitive applications. Are the current development tools able to keep pace with this development?

In the coming years we will experience more and more technical innovation in the field of automotives, which will be mostly attributable to the use of microcontrollers, mechatronics and the relevant software. In the course of this, new comfort functions as well as security-sensitive functions like driver assistance system will have growing importance in our daily life. Active and passive safety can be increased through the help of assistants that help to realize driver input (e.g. ABS, ESP) during normal drive (e.g. ACC), in situations of potential danger (e.g. LDW), in acute danger (e.g. Emergency Break System), as well as during and after accidents.

One the one hand, driver assistance systems increase road safety, but on the other hand they also bear a potential of new dangers due to malfunction. Here, with its strict guidelines, the automotive industry tries to provide the best possible operational reliability and a failure rate as low as possible.

IEC 61508 [1] is a basic safety standard for these fields. However, this standard is kept very general and needs further detailing according to the fields of operation. The aviation industry was a pioneer in this respect, since it defined a very concrete safety requirement with the so-called standard RTCA DO-178B [2]. This standard divides the software into five certification levels ("Software Level", table 1) according to the consequential damages of a failure or a malfunction. At the same time, it requires proof for "Code Coverages" of different complexity according to the certification level.
These range from a simple review of the requirement to the so-called "Statement Coverage", where each command of the source code has to be executed once, to the "Modified Condition/Decision Coverage", where even each possible partial condition of a conditional command for "true" and "false" has to be run.

Level	Danger Level	Required Code Coverage
A	disastrous	like Level B, additionally: Modified Condition / Decision Coverage (MC/DC)
B	dangerous / severe	like Level C, additionally: Branch/Decision Coverage
C	significant	like Level D, additionally: Statement Coverage
D	minor	Coverage of the requirments
E	no impact	no specific requirements

Table 1: Required Code Coverage for certain danger levels (acc. To DO-178B [2])

In the field of automotives, people are also working at adequate concretions and adaptations of the safety requirement, amongst others standard ISO WD 26262 [3] as an "automotive" derivative of the IEC 61508. Furthermore, also AUTOSAR [4] or the EU's project "EASIS - Electronic Architecture and System Engineering for Integrated Safety Systems" [5] have to be mentioned.

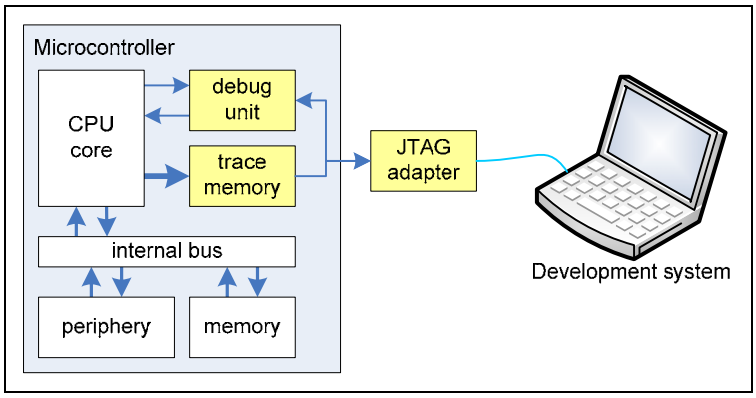
Security needs Trace

Along with sophisticated tools for a clean generation of software, the subsequent test of the produced software is of great importance. Trace data, which contain information on the program executed on a microcontroller, are an important basis of these tests. A simple trace contains instructions executed by the CPU. For advanced analyses, like for example the "Modified Condition/Decision Coverage" mentioned earlier, additionally, data read by the CPU is necessary. Furthermore, information on data written by the CPU as well as data modified during a DMA transfer is important.

To continue the discussion on verification of Code Coverage in security-sensitive applications, a short overview will be given on the techniques that can give a complete trace (at least locally limited). In this context, a discussion will be given in detail on how much does a technique influence the system, how realistic is the timing information of the extracted trace data and how exactly do the data correspond to a program run on an actual series processor. Moreover, the time and effort necessary to supply a relevant trace tool for a new processor type are relevant, of course.

At present, the most efficient, but also most expensive technique to extract trace data, is the implementation of In-Circuit-Emulators (ICE). In this case, special versions of the chips are produced that permit access to all relevant internal signals via dedicated pins (“Bond-Out Chips”). This also permits for example to record special register contents in real-time. Due to the enormous costs (often more than EUR 10,000) and the delay between the availability of samples and that of emulators, this technique is used less and less frequently by the industry.

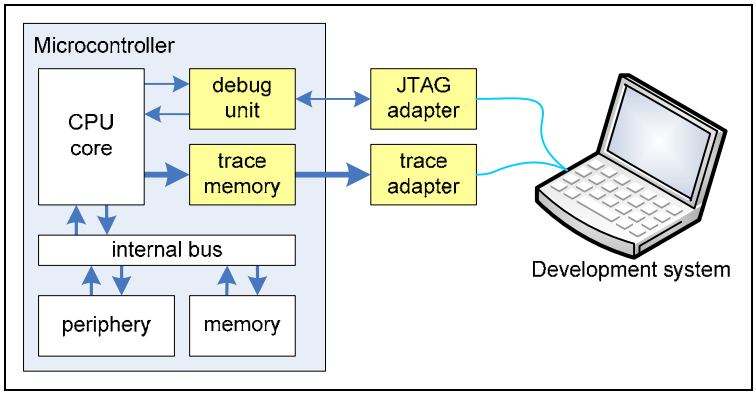
Also, the use of such Bond-Out Chips is limited to frequencies up to 50 MHz, since at higher frequencies the signal quality is influenced so badly by the wiring, that malfunctions happen (e.g. DDR SDRAM Interface).



Picture 1: Collection of trace data by using a microcontroller’s existing debug abilities combined with a local memory for trace data

Meanwhile, in order to extract trace data, modern processors often also contain a simple hardware support that will also be included in the serial chips. The simplest type of that is shown on [picture1]. By including a trace memory on the chip, relevant internal data can be written into this memory. Usually, the trace memory is organised as a ring memory, so that always the most recent past is stored. Mostly, the addresses of the instructions executed are stored. Now, this buffer can be readout through the JTAG interface, whereby the bandwidth of the readout is limited so strongly that usually it is not possible to work continuously. In order to still cover a maximum program area, possibly, there is the opportunity to filter the data, written to the trace buffer, so that only the subsequent addresses of jumps will be

recorded, for example. For this technique additional investment in chip surface for the trace buffer and the filter logic has to be made. Since this technique is mostly used in serial chips, record depths are strongly limited for reasons of economy. This results in the fact that the microcontroller that is to be tested can process a few hundred instructions and has to be stopped afterwards to readout the trace data.



Picture 2: Collection of trace data by using a microcontroller's existing debug abilities combined with a local trace memory and a separate interface for trace data output

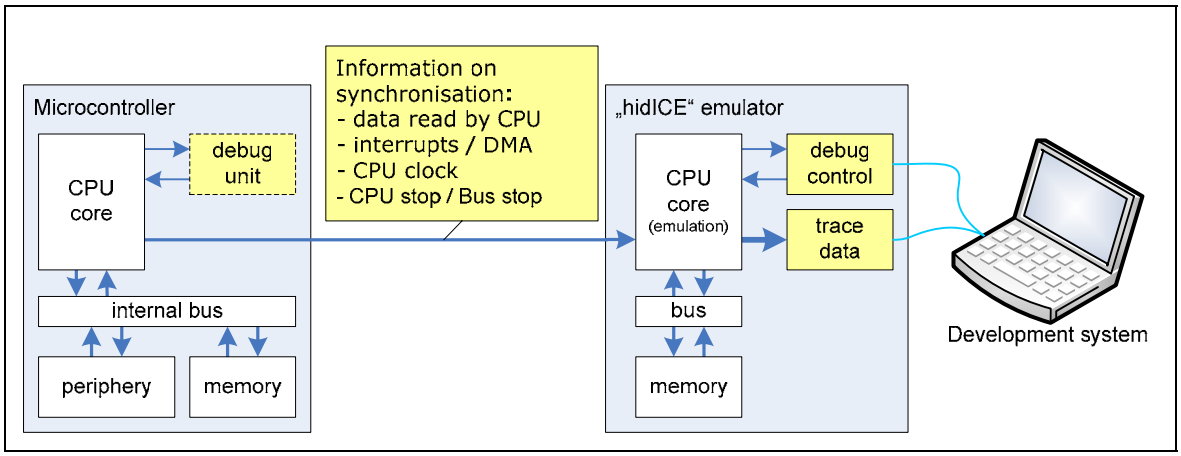
An improvement can be reached by not using the JTAG interface for the outward transport of trace data, but by implementing a special trace bus on the chip (frequently with 4, 8, or 16 bit width). Through sufficiently high clock rates on this bus (often several hundred MHz) it is possible to transmit the recorded trace data outward [picture 2]. This external trace application with a special trace adapter can record almost arbitrarily deep. But still you have to weigh up the complexity of the hardware against the records' detailing level. With this method you also have to accept the halt of the microcontroller to avoid gaps in the recorded trace.

Efforts for standardisation

As the debug/trace problem concerns of course all producers of semi-conductors, there have been efforts for standardisation in this field. This results in the so-called NEXUS Forum [6]. NEXUS is actually an abbreviated designation for Standard IEEE-ISTO 5001. This standard aims at being able to debug processors of different producers with the same tools. It provides four different classes of debugging. These range from simple operation control (breakpoints, reading/manipulating of registers and storage is designated as class 1) to extensive traces and advanced operation control (real-time traces, write-data traces, triggering through watch points, processor halt at trace buffer spillover is designated as class 4).
Being aware of the complexity, the NEXUS Standard explicitly disclaims a demand for a simultaneous record of data and program trace, as well as for a trace of DMA accesses.

More trace information than permitted by NEXUS

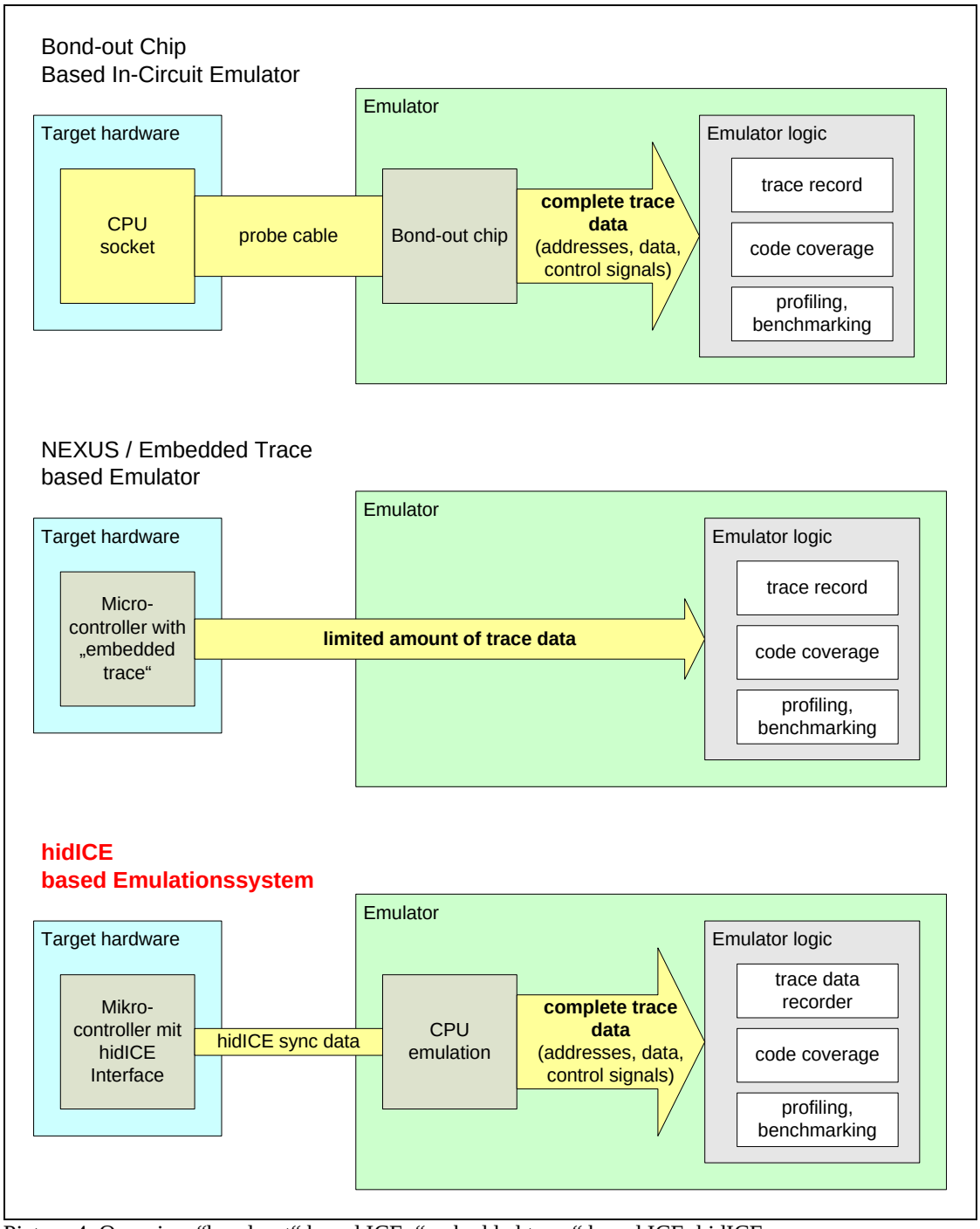
This is where Accemic's hidICE technology comes into play. HidICe is a surprisingly easy way to supply essentially more trace information than defined in the NEXUS Standard. This excess of trace data is combined with a comparatively low effort to extract trace data from a microcontroller.



Picture 3: Principle of “hidICE“ emulation

The basic principle of hidICE technology is that the trace data are not directly extracted from the target microcontroller, but from a paralleling emulation of the CPU. The second CPU is located directly in the emulator. Of course, the CPU must have the same functions like the CPU of the microcontroller that is to be observed. If the program storage of both CPUs contains identical codes, both CPUs will execute identical instructions after a reset initial. The individual program run occurs only by data that are read and analysed during runtime of the program and by the influence of occurring events (interrupts). The idea behind hidICE is that the data necessary to synchronize both program runs are transferred from the target microcontroller to the emulated CPU. Thus, you can make sure that both CPUs experience the same “external” influences and therefore, have the same program sequence. Then it is easy to access all trace information within the emulator, whereby not only executed instructions and all date read and written can be accessed, but along the way also further important information like addresses and data transferred during DMA transfers are available, as well as a trace of the CPU registers.

In the embedded-trace solutions discussed before, skips and data read and written by the CPU are transported with lots of effort. In doing so a lot of engineering is employed upon maximizing the efficiency of transferring these data and upon minimizing data losses and the influence on the system through necessary halts of the CPU respectively.
It is easy to see at this point that the synchronisation of both CPUs in a hidICE system represents only a percentage of the data volume that has to be transferred in an embedded-trace solution.
To synchronize a hidICE interface, basically only information on an interrupt occurred, on temporary halts of the bus and the CPU, as well as data read from the IO-area (UART, CAN, ADC, etc.) have to be transferred. If this information is transferred synchronously with the CPU clock, it is very easy to reconstruct the complete program sequence in the CPU being emulated.



Picture 4: Overview “bond-out“ based ICE, “embedded trace“ based ICE, hidICE

An attentive reader might wonder now what would happen if a fault occurred in the emulation. Then, both CPUs would drift apart inevitably and the extracted trace would be useless. In order to recognize such faults, the target CPU will continuously make checksums of data read and written, instructions executed and addresses of instructions and data. If this checksum is the same as in the CPU being emulated, one can state if both CPUs run synchronously. In the process the checksums can be transferred while there are no data read to be transferred. Not only is it possible to monitor the correct functioning of the emulation with this method (hidICE System Integrity Control), but simultaneously also sporadic faults can be recognized that occur in the target CPU. Reasons for these sporadic faults might be instable storage cells, problems with the voltage supply, too narrow timings, etc., which cannot be found with other methods, or only with lots of effort. HidICE technology can recognize

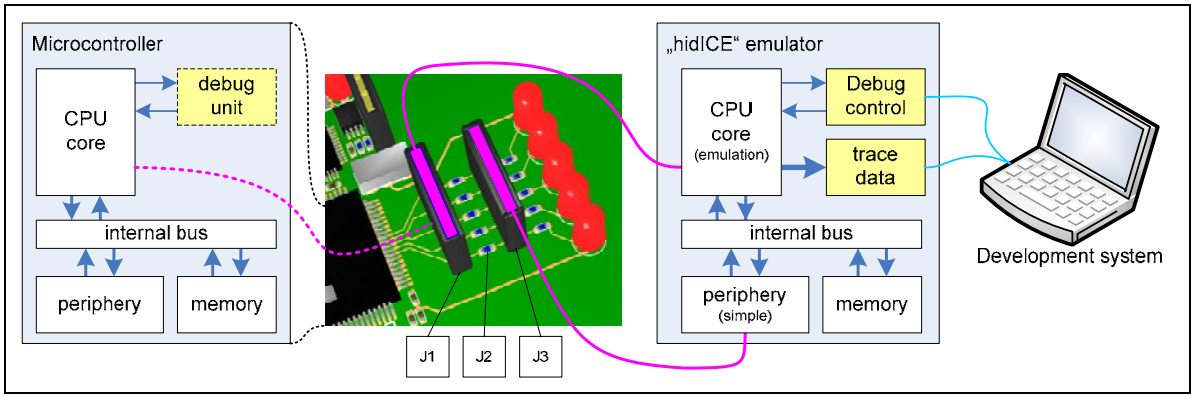
exactly the moment when such a problem arises, and additionally, a trace with the fault's history is available.

Any number of break points can be implemented on the emulator. Thus, the elaborate logic for break points in the target CPU does not apply here. Due to the required transmission time of data read, the emulation is increased by a defined number of CPU cycles. For this reason, at the occurrence of a valid break condition in the emulator, the target CPU will halt a few clock cycles later. For most break points this is acceptable, but additionally, break points can be implemented in the target CPU as usual so far, and those will be able to halt the target CPU without delay. We recommend using break points on the emulator to cover fault conditions, which actually should not occur, and to use the break points of the target CPU for direct program control (step, goto, etc.).

On the target CPU the functionality of the hidICE interface is limited to the "collection" of information relevant for synchronisation (primarily CPU clock, data read, interrupts, DMA requests, and bus and pipeline stop). Therefore, the effort for implementing a hidICE interface is surprisingly small. You can assume less than 1000 gates for a 32 Bit CPU.
The number of I/O pins that is necessary to output the sync information depends on the bus amplitude of a read access on the I/O area as well as on the number of clock cycles needed by the CPU to do the read operation. But in any case, the number of necessary I/O pins is essentially lower than for a NEXUS interface with comparable efficiency. For some established 16 bit MCUs 5-7 pins are necessary, for 32 bit MUCs that are 10-14 pins. These figures refer to exemplary implementations until now and can vary from CPU to CPU.

Additionally, one can also use those pins together with the JTAG interface that usually exists anyhow. Mind you that with this little effort it is possible to access a simultaneous real-time trace of instructions, data read and written, DMA and CPU registers – this is a functionality that by far exceeds the functional range (required and optional) defined in NEXUS level 4.

With the mechanism for port replacement also defined by NEXUS, the number of I/O pins necessary to output the sync information can be reduced to zero, provided these pins are used for simple output functions [picture 5].

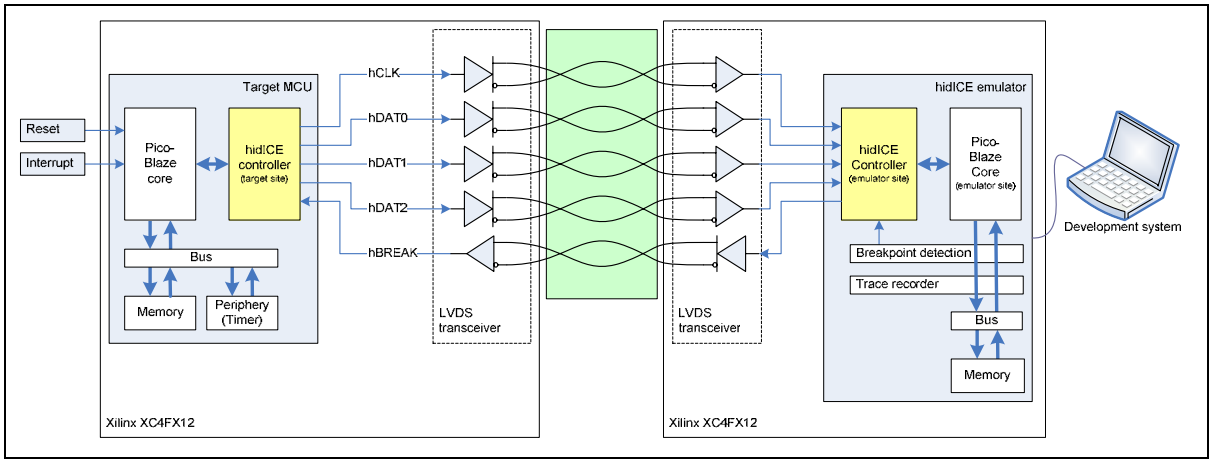


Picture 5: Port Replacement: If the hidICE emulator is connected, the information necessary for synchronisation of the CPU cores will be transferred to the emulator through J1. J2 is open. The output of the periphery being emulated will be re-imported to target hardware through J3. If the hidICE emulator is not connected, the J2 jumpers will be connected and thus, the microcontroller can directly address the periphery.

As the effort to extract trace data is that little, there is no obstacle to use the hidICE interface in serial microcontrollers. Thus, no more bond-out or special evaluation chips have to be used and the software can be developed and certified on the same microcontroller that will be used

in mass production. This is an invaluable advantage, since it gets increasingly more difficult for producers of semiconductors to guarantee an absolutely identical performance of evaluation chips and mass production chips. But if this consistency is not given, a good portion of the test effort for a new device is made absurd, since in the end a device that is supplied with mass production might show a completely different performance in the field than the device tested before with the evaluation chip.

The operability of the hidICE concept can be verified with a hidICE demonstration system. It consists of 2 Xilinx Virtex4 Boards that are connected with each other via 5 LVDS lines. As CPU we chose the Xilinx PicoBlaze implementation, since conveniently, it is available completely in Verilog. The system is clocked with 200 MHz and supports asynchronous events (reset and interrupt). The hidICE demonstration system permits to record a trace, and to set break points (in the target CPU and in the emulated CPU). With an infiltrated defective instruction, the demonstration also shows the reliable functionality of the hidICE system integrity control.



Picture 6: hidICE demonstration system

Recapitulating, the hidICE technology offers a functionality that essentially exceeds NEXUS level 4. HidICE combines the advantages of both established trace techniques by combining the complete trace data that can be extracted from a bond-out chip with the high clock frequencies of an embedded-trace solution. The implementation of a hidICE interface in a microcontroller can be a lot more cost-efficient than an embedded-trace solution. HidICE can be used in mass production MCUs without a lot of extra costs, and it permits the development and certification of application software under real conditions on the same hardware that will be later used in the field.
Thus, hidICE is a surprisingly simple and cost-efficient solution that allows certification of application software on mass production MCUs under most difficult test conditions ("Modified Condition/Decision Coverage")

Sources

[1]	IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems. IEC 1998.
[2]	Software Considerations in Airborne System and Equipment Certification- RTCA/DO-178B. RTCA Inc., Washington D.C., December 1992.
[3]	ISO/WD 26262
[4]	www.autosar.org
[5]	www.easis.org
[6]	www.nexus5001.org

Editor background:

Prof. Dr. Christian Hochberger
holds a masters degree and PhD in computer science from the technical university of darmstadt. After spending two years as a freelance developer in several industrial projects he was appointed assistant professor at the university of rostock in 1999. Currently he holds the chair for embedded systems at the Dresden university of technology. His research interests include reconfigurable embedded systems, Java in embedded systems and debugging of embedded systems.

Dipl.-Ing. Alexander Weiss
is managing director and co-founder of Accemic, a Munich area based company for innovative microcontroller development tools. After receiving his master degree (Dipl.-Ing.) in electrical engineering from the Technical University of Munich in 1996 he gets scientific assistant at the Munich University, where he designed safety-critical applications in the region of biomedical engineering.
At Accemic he provides technical direction for the design of software development tools for embedded systems and exploring and developing new debugging technologies."