

This paper was presented during the EtroniX conference, held Feb. 25th – March 1st 2001 at the Anaheim Center, Anaheim, California.

USING MICROPROCESSOR AND DSP DEBUG INTERFACES FOR MANUFACTURING FUNCTIONAL TEST AND DIAGNOSIS

Billy Fenton,
Vice President of Engineering,
International Test Technologies.

Abstract

As board test access begins to disappear, industry roadmaps [1] predict a move from traditional in-circuit test methods to automated inspection techniques, JTAG, and functional test. Although requiring minimal test access, functional testers have a number of shortcomings, namely: long development times, reduced diagnostic resolution, and long test times.

Many microprocessors and DSP manufacturers now include debug interfaces, which are intended to simplify board hardware and software development. Although aimed at design engineers, test engineers can exploit these interfaces to implement functional test solutions, with reduced development times, improved diagnostic resolution, and shorter test times.

Combined with automated inspection and JTAG, functional testers using a debug interface, provide a good alternative for present and future test strategies. This paper will discuss how an alternative test technique such as the one in this paper improves test development, improving time-to-market in both manufacturing and field service.

Defining Functional Test

Test is an important part of the manufacturing cycle, accounting for 33% or more of manufacturing cost [1][2]. Typically, it consists of both a structural and a functional test. A structural test verifies that a board was built correctly, either using an In-Circuit Tester (ICT) or JTAG, whereas a functional Test verifies a board's design and performance by mimicking its behavior in a target system. The functional test strategy employed for a particular product will vary and a number of possible options are reviewed in the next section.

A Review of Traditional Functional Test Methods

This paper is primarily concerned with methods for functionally testing processor-based boards. The structure of both non-processor and processor boards is shown in Fig. 1. The functional test method used in a particular application is often influenced by the board structure.

A non-processor board can be treated like a 'black box'; its circuitry simply translates its input signals to a new output form. Stimulus/response methods are often the best test approach. Processor boards have a more complex tree-like structure, and involve a tight integration between hardware and software. Therefore, traditional stimulus/responses test methods are generally not appropriate.

Approaches to functional test can be categorized as follows, and each is briefly reviewed in the next few sections [4].

- Digital Stimulus and Measurement
- 'Rack and Stack' or Virtual Instruments
- Hot Mock-Up
- Built-in Test (BIT)
- Emulation Testing

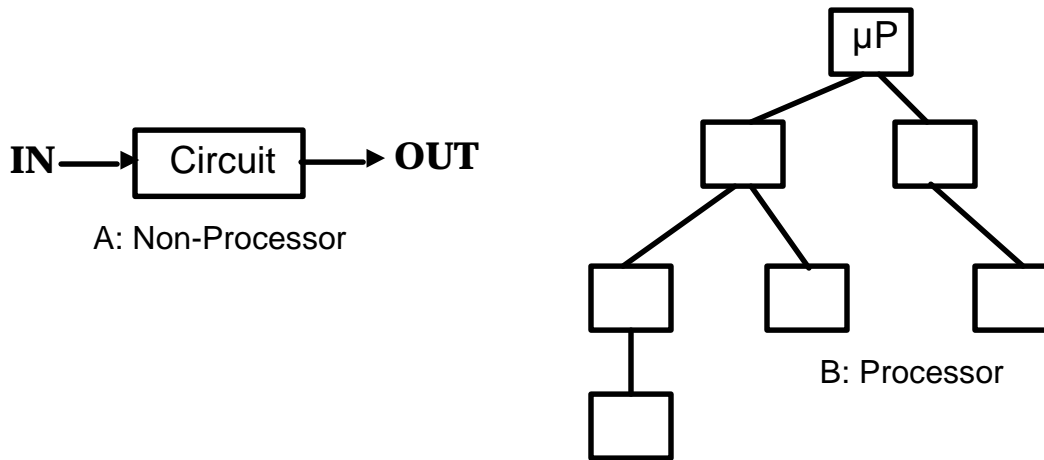


Fig. 1: Board Structures

Digital Stimulus and Measurement

The output of a digital circuit is defined by the binary patterns on its current and/or previous inputs. Therefore, a circuit of the type shown in Fig. 1A can be tested by applying a series of test patterns or vectors to its inputs, and measuring the responses at the outputs. A correct set of responses verifies the circuit [3]. Fig. 2 shows an example.

Advantages are:

- CAD software can be used to automatically generate the test and response vectors.
- Diagnosis can be automated by using fault dictionaries. A fault dictionary defines the behavior of the test and response vectors during the presence of certain faults, therefore if a test vector returns the incorrect response, the fault dictionary returns the fault(s) associated with this incorrect response.

Disadvantages are:

- The required amount of test and response vectors becomes intractable for large circuits, such as processor-based boards.
- Only suitable for digital circuits.

'Rack and Stack' or Virtual Instruments

This approach integrates a range of instruments through a common programming platform or test executive. The instruments are used to stimulate and measure the Unit-Under-Test (UUT), using analog and/or digital signals. Typically, IEEE-488, PXI, or VXI instruments are used [4]. Fig. 3 shows a simple example.

Advantages are:

- It can be used for both analog and digital test.

- There is a wide range of instrument options, making it a suitable candidate for almost all forms of electronic test.

Disadvantages are:

- Test development times can be long.

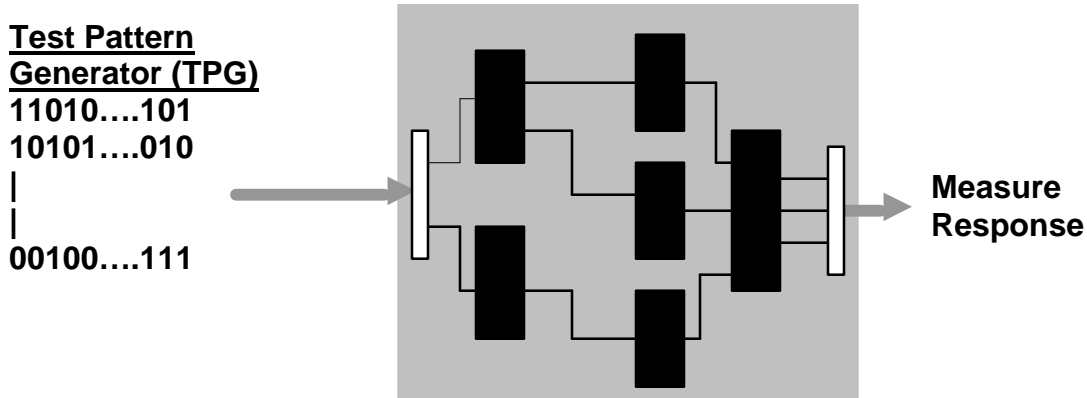


Fig. 2: Digital Stimulus/Response Testing

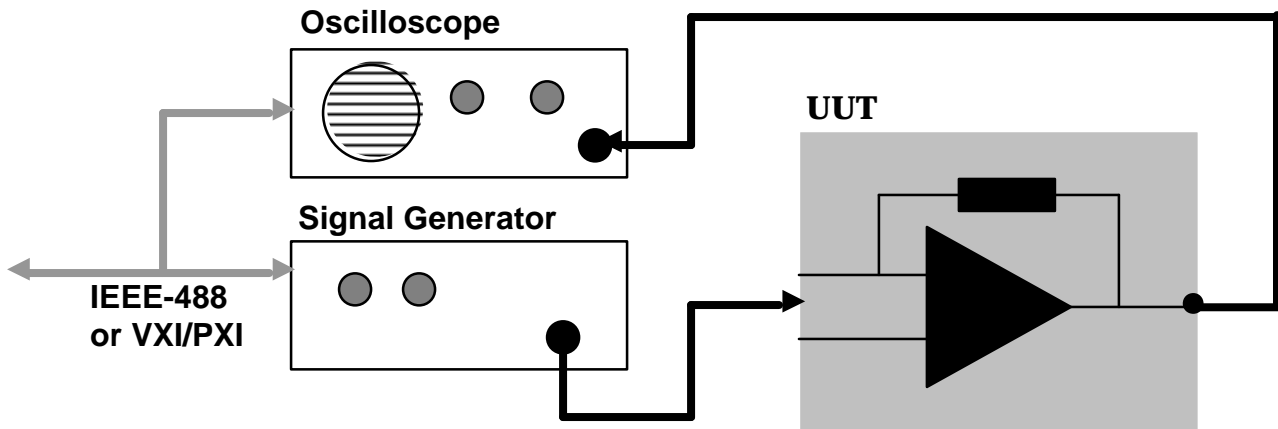


Fig. 3: 'Rack and Stack'

Built-in Test (BIT)

At design time, some hardware and/or software test mechanisms are included. At power-up or on receiving a prompt from an internal or external source, the board verifies its own operation. Methods include:

- Chip level BIST – Extra hardware is included at the chip level to perform stimulus/response testing. Initiation is either on power-up, or via JTAG.
- Power-on Self Test (POST) – The boot ROM or a specially loaded test ROM perform diagnostics. Error reporting is often via a serial port connection.
- Software Diagnostics – After initial boot, diagnostics are loaded and run from disk.

Advantages are:

- Often implemented by the designer, which removes the burden of test engineering from manufacturing.

Disadvantages are:

- Completely 'dead' boards cannot be diagnosed.
- Design For Test (DFT) is required.
- Post design modifications to test procedures can be difficult.

Hot Mock-Up

A hot mock-up is a real system, minus the board to be tested. During test, the UUT is plugged into the hot mock-up and self-tests or other tests specifically designed for the situation are run.

Advantages are:

- Initial set-up cost is low.
- In many cases test programs already exist, so test development times can be low.

Disadvantages

- Very labor intensive, as fixturing is typically manual.
- Test times can be long as the boards operating system will have to boot.
- Testing is go/nogo and diagnostics can be limited.

Emulation Testing

Emulation testing is used with bus-based boards. The emulator typically replaces some part of the on-board circuitry, and takes control of the board via one of its buses. Once in control, the emulator can load and run tests and diagnostics on the UUT.

Types of emulation include:

- Processor
- ROM
- Bus

A processor emulator replaces the board's processor, and gives full control over all the processor buses. Once in control, full read/write access to all parts of the UUT's memory and I/O is available via the emulator. It provides the ideal method of test access for processor boards. However, processor emulation is not viable on higher speed processors (>20-30MHz), and the real processor must be socketed, so that it can be easily replaced by the emulator.

A ROM emulator replaces the boot ROM, and substitutes diagnostic code for the processors normal boot code. In addition, a ROM emulator has a bi-directional communications pathway, so that the UUT can communicate with the tester. Its primary shortcoming is its inability to diagnose the processor to boot ROM area. In addition, it can be difficult to use with soldered-in ROMs unless some circuit modifications are made at the product design stage.

A bus emulator connects to a bus slot or edge connect, and provides the ability to perform read/write bus cycles on the UUT, thus providing test access to the various UUT circuits and functions. It's a good solution for testing plug-in bus cards (e.g. VME, PCI, etc.).

Emulation was a popular approach in the 80's and early 90's; its decline was driven by the absence of socketed processors and ROMs in new designs, and by increasing processor speeds, which made emulators difficult or impossible to design. However, with the advent of processor debug interfaces, it's now time to reconsider this powerful approach to functional test and diagnosis.

What is a Microprocessor/DSP Debug Interface?

Designers require test tools, such as processor emulators and logic analyzers, for debugging both hardware and software. However, it is difficult to design emulators for high speed processors (>20MHz), and most modern processors operate at frequencies far in excess of this. Processor vendors realized this and solved the problem by incorporating the required debug functions on the processor die. In essence, a 'processor emulator' is now inside the processor!

Typically, the debug functions incorporated include the following low-level functions:

- Stop the Processor
- Read/Write Memory
- Read/Write I/O
- Breakpoints
- Single Step Code
- Code Trace

Access to the debug functions is typically via the processors JTAG port, plus 2-3 special purpose signals. Processor designers simply extend the JTAG instruction set to include vendor-specific instructions for controlling the processor core.

How can the debug interface be used for manufacturing test and diagnosis? The debug interface allows the tester to take control of the processor, and then the processor is simply used as a 'vehicle' for testing the rest of the UUT. For example, the debug interfaces read/write functions can be used to perform memory testing.

Some Debug Interface Examples

Many modern microprocessors and DSPs now include a debug interface. Some examples are:

- Intel's Pentium Family
- Motorola's and IBM's PowerPC Family
- ARM [5]
- MIPS
- SPARC
- Motorola DSP
- AMD Elan [7]

Additionally, the IEEE recently published a standard (IEEE-ISTO 5001) for debug interfaces [6].

Using a Debug Interface for Functional Test

Fig. 4 shows an example processor board. The 'Debug Interface' Tester connects to the processors debug port. This connection requires about 5-8 test points on the UUT. The tester takes control of the processor before it runs any code, using the debug interface. Once in control of the processor, the tester simply uses the processor as a 'vehicle' for testing the rest of the board. Typically, the tester can perform the following low-level functions via the debug interface:

- Start/stop the processor
- Read/Write Memory
- Read/Write I/O

- Breakpoints
- Download test code to the UUT memory
- Control execution of test code in the UUT memory
- Collect test results from UUT memory

A UUT can be divided into functional blocks. For example, the UUT in Fig. 4 contains the following functional blocks: processor, north bridge, SDRAM, video controller, etc. Each functional block contains arrays of memory or I/O registers. These registers control the operation of the functional blocks, and are accessed using the debug interface low-level functions. Therefore the debug interface low-level functions are used to build up a test program for each functional block, and these test programs can then be sequenced to create a complete UUT test. Some example test programs are discussed in the next sections.

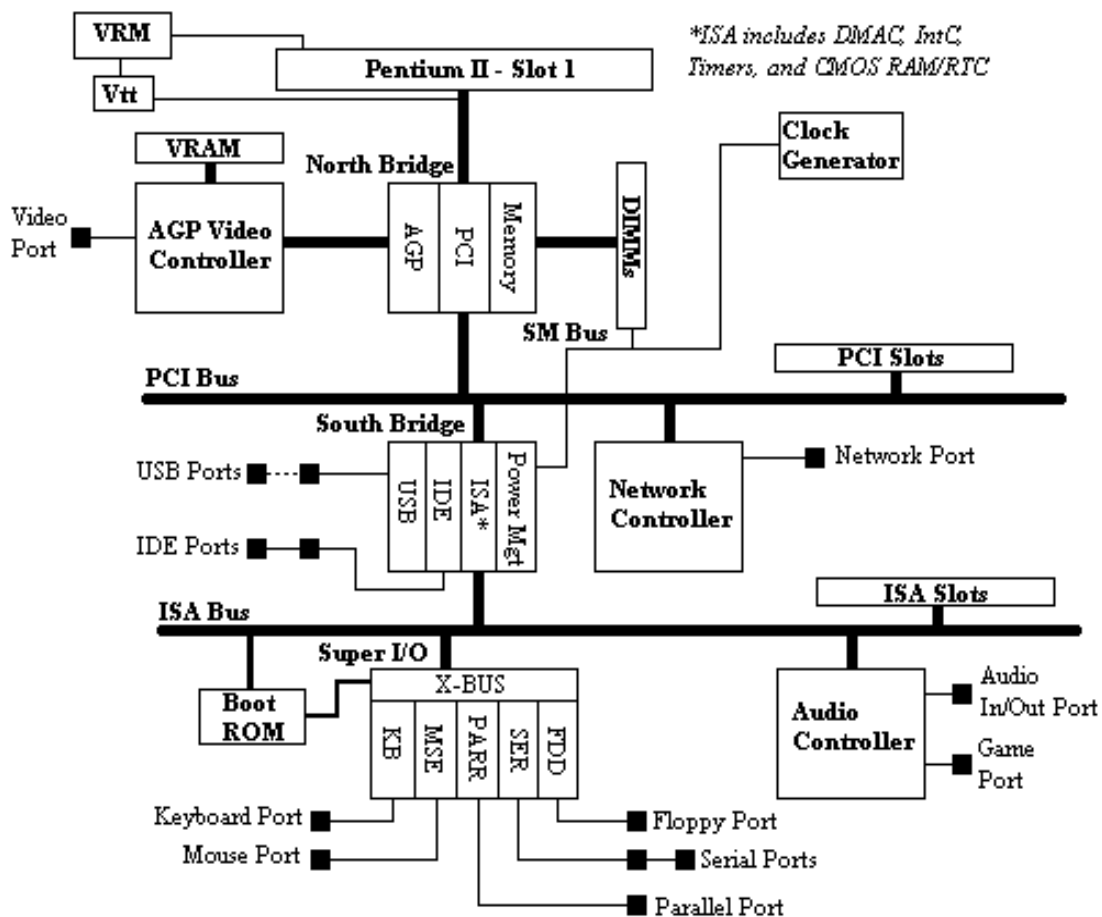


Fig 4: Example Board

Example 1: Memory Test

The debug interface low-level functions can be sequenced in the following way to create a simple memory test.

- Program the NorthBridge registers using the debug interface Write I/O function.
- Write AAAAAAA to memory location 0, using the debug interface Write Memory function.
- Read and verify memory location 0 for AAAAAAA, using the debug interface Read Memory function.

- Write 55555555 to memory location 0.
- Read and verify memory location 0 for 55555555

Obviously, more complex memory test algorithms can be implemented using the same approach, and many commercial testers are supplied with pre-programmed memory tests.

Example 2: Serial Port Test

The debug interface low-level functions can be sequenced in the following way to create a simple memory test. A loopback plug is used for verifying the operation of the serial port.

- Program the serial controllers control registers for required bit rate, number of data bits etc., using the debug interface Write I/O function.
- Transmit a character by writing it to the serial controllers transmit register, using the debug interface Write I/O function.
- Verify transmitted character is received via the loopback, by reading the serial controllers receive register, using the debug interface Read I/O function.

Obviously, more complex I/O devices can be implemented using the same approach, and many commercial testers are supplied with pre-programmed tests for many common I/O components and standards including: Video, USB, IDE, Bridges, Audio, etc.

“But, most modern board require very complex test sequences?”

That’s true, but most commercial testers are supplied with utilities and libraries to overcome the complexity of programming these devices.

Device libraries are available for the most common chipsets and devices. Import these into your test sequence, and you are ready to test.

In addition, learn utilities monitor the boards own initialization procedures and automatically generate test programs to mimic it. For example, use a video driver to initialize a video chip, then use the learn utility to copy this initialization procedure.

Using a Debug Interface with Other Instruments

By using the debug interface on its own, an I/O controllers ability to communicate with the outside world is not verified. A loopback or a real peripheral can be used in many cases. However, the other alternative is to combine the ‘Debug Interface’ tester with other instruments. ‘Debug Interface’ testers are supplied in many form factors including PCI and PXI, and are supplied with DLL instrument drivers. Therefore, using the approach with other instruments is relatively straightforward.

Fig. 5 shows the set-up for testing a simple mixed-signal boards. Table 1 shows a test sequence for this board.

Diagnostics Included!

A debug interface tester takes control of a processor before it fetches any code. This means that even defective bus lines at the processor can be detected and isolated automatically! Therefore, that bonapile of ‘dead’ boards can be a thing of the past.

Additionally, if tests are sequenced correctly, even go/nogo tests can provide useful diagnostic information. Tests are arranged and executed in a top-down fashion, starting with the processor. Therefore, the first failure will clearly isolate the defective block.

Also, the test results provide detailed diagnostic information such as the location of open or shorted bus lines.

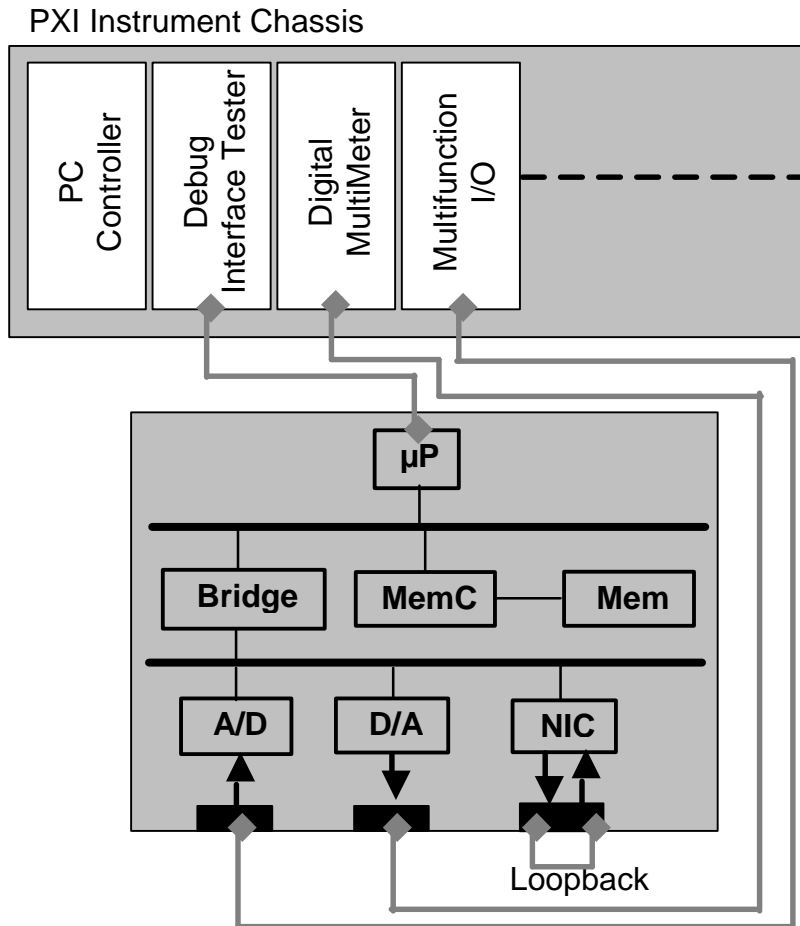


Fig. 5: The Debug Interface and Other Instruments

UUT Function	Test
µP	'Debug Interface' tester takes control.
MemC (Memory Controller)	Register Test Configure Registers.
Mem (Memory)	Memory Test.
Bridge	Register Test Configure Registers.
A/D	Generate voltages using multi-function I/O instrument. Read from A/D register using the debug interface Read I/O command. Verify result.
D/A	Generate D/A voltage by writing value to D/A register using the debug interface Write I/O command. Measure D/A voltage using the DMM. Verify Result.
NIC (Network Interface)	Register test.

Controller)	Transmit and receive packet via loopback.
-------------	---

Table 1: Test Sequence

Other Issues

Commercial debug interface testers are available in common form factors such as PCI and PXI. This means that it is possible to integrate these tools with many functional testers and automated board handling solutions.

Also, compatibility with industry standard test executives means that connection into your wider enterprise via networks, databases and SPC, is straight forward.

Hybrid Approaches

It is not claimed that a debug interface tester is a panacea to all functional test ills. It can either be used as the 'core' of a functional test solution, or as a complement to existing approaches. Such hybrid approaches might include:

- For products with limited test access, it can be used as a complement to in-circuit test. Use the debug interface tester to test the sections of the board not accessible by the ICT.
- It expands the test coverage capability of conventional JTAG.
- Use to verify sections of the boards, which BIT cannot reach.

Benefits of Using a Debug Interface for Functional Test

Debug interface solutions provide a number of significant benefits over existing functional test solutions. These include:

1. Test times can be seconds rather than minutes, leading to substantial test cost savings.
2. It's not just go/nogo, full diagnosis is provided by default.
3. Test programming is under full control of the manufacturing team, so reliance on a busy product design department for test program changes should be a thing of the past.
4. Commercial solutions come supplied with a comprehensive device library and learn utilities, so test development times are greatly shortened.

Conclusions

Emulation provides a useful tool in your functional test armory. Popular in the 80's and early 90's its use faded because of increasing processor speeds and product miniaturization. However, with the advent of processor debug interfaces, its use is again possible.

Using 5-8 test points on the UUT, the debug interface stops the processor. Once in control, the processor is simply used by the debug interface to access and test all board functions.

By bypassing the board's normal operating software, functional test times can be slashed from minutes to seconds. Additionally, commercial testers are supplied with pre-programmed device libraries and learn utilities, thus greatly reducing test development effort. Lastly, the approach provides comprehensive diagnostic information, which will make bonepiles of 'dead' boards a past memory.

References

1. NEMI, National Electronics Manufacturing Technology Roadmaps, 1998.
2. Bernard Sutton, Board Test and Product Lifecycle, IEEE Design and Test of Computers, July-September 1999, pp. 28-33.
3. Samiha Mourad & Yervant Zorian, Principles of Testing Electronic Systems, John Wiley & Sons, 2000.
4. Stephen F. Schreiber, Building a Successful Board-Test Strategy, Butterworth-Heinemann 1995.
5. Application Note 28: The ARMTDMI Debug Architecture, ARM Ltd., 1995.
6. Global Embedded Processor Debug Interface Standard (IEEE-ISTO 5001), www.NEXUS-STANDARD.org.
7. Daniel Mann, AMDDebug offers on-chip Debug Support, Embedded Systems Engineering, Dec/Jan 2000.