

# HARDWARE VALIDATION

*By Clive (Max) Maxfield and Joe Skazinski*

HARDWARE VALIDATION



## Only 10 Days to Shipping ... We May Have a Memory Problem!

*How can software development engineers be provided a fully validated hardware platform upon which they can perform their final application integration prior to customer shipment?*

How many times have you been in a situation working with a new system where the board bring-up occurred without any major problems manifesting themselves (hey, it could happen)? Initially, everything seems to work just fine. The rudimentary diagnostic tests provided by the hardware guys indicate that all is as it should be. The application software appears to be working as planned. The customer ship date is fast approaching. Everyone on the team is starting to feel confident... but suddenly... with only a few days to go... everything grinds to a halt.

Possibly an application is attempting some new task for the first time – perhaps a DMA transfer with the CPU cache disabled. But why is the system crashing? Surely someone must have validated this mode of operation... didn't they?

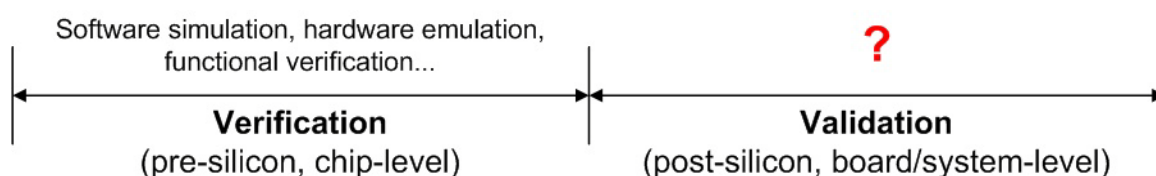
**Contents**

Verification and Validation.....4  
Early Board Test Techniques.....4  
Rudimentary Hardware Validation Techniques.....5  
Hardware Validation TNG (The Next Generation) .....6  
Bios .....10

## Verification and Validation

For the purposes of this paper, the term "verification" is understood to refer to any pre-silicon activities that are performed to ensure that individual silicon chips will function as planned. By comparison, the term "validation" is taken to refer to any post-silicon board/system-level activities that are performed to ensure that the system as a whole, complies with original requirements.

The creators of modern electronic systems are currently faced with something of a dichotomy. During the front-end of the product development process while designing any custom devices – such as System-on-Chip (SoC) components, for example – engineers have access to a wide variety of verification tools and techniques, such as software simulation, hardware emulation, and formal verification, to name but a few (Figure 1).



**Figure 1. Verification versus Validation**

The problem arises when these custom-designed components are brought together with other devices to form what is typically an extremely complex system. Validating the system as a whole prior to installing and integrating any application software is particularly important when considered in the context of today's ever-shrinking market windows. To maximize market opportunity, a project must minimize the amount of time spent validating the system without negatively impacting the quality of the validation.

Product development teams simply cannot afford to waste time trying to work out whether a problem resides in the hardware or the software. It is necessary to provide the software development engineers with a fully validated hardware platform upon which then can perform their final application installation and integration prior to customer shipment. But just how is the system hardware going to be validated prior to any application software being loaded, run, and – if necessary – debugged?

### Early Board Test Techniques

In the not-so-distant past (circa the 1980s and 1990s) there were several techniques that were widely used to test assembled printed circuit boards (PCBs). One approach employed something known as a "bed-of-nails" tester, which involved a test bed covered with an array of spring-loaded pins. The circuit board to be tested was pressed down onto the pins, each of which made contact with a given pad or test point on the board. The other side of the pins was connected to the circuitry that was used to apply the tests to the board and monitor the

results. In addition to being horrendously expensive, bed-of-nails testers were mainly applicable to boards with components on only one side. Also, the tests typically involved isolating the components (using special pins to drive surrounding devices into a high-impedance condition) and testing them in isolation, which therefore did little to validate the board as a whole.

Another common approach was known as "functional test". In this case, the circuit board was plugged into a functional tester by means of its edge connector. The tester then applied a pattern of signals to the board's inputs, clocked those values into the board, allowed sufficient time for any effects to propagate around the board, compared the actual values seen on the outputs with a set of expected values stored in the system, and repeated the process for thousands (sometimes tens-of-thousands) of additional test patterns. One disadvantage of this approach was that the board was not run at full speed; another issue was that the tests really verified only the most basic hardware functions, but they did not validate the board as a whole.

### *Rudimentary Hardware Validation Techniques*

When it comes to validating the system, perhaps the most primitive level is to create a rudimentary assembly program, compile it, load the machine code into the system's memory at the default boot address, and trigger a system reset. Upon being reset, the CPU executes this machine code program performing simple pre-defined tests. (Don't laugh. The authors of this paper have both used this technique in the past.)

The next step up the evolutionary ladder that is often used is to add test functions to a boot loader. After the modified boot loader is compiled, loaded into memory, and the system is reset, the user can execute pre-defined rudimentary tests.

Another level of sophistication is to create a special validation application in a higher-level programming language like C/C++, compile this application into machine code, load the machine code into the memory at the default boot address, and – once again – trigger a system reset to run your application.

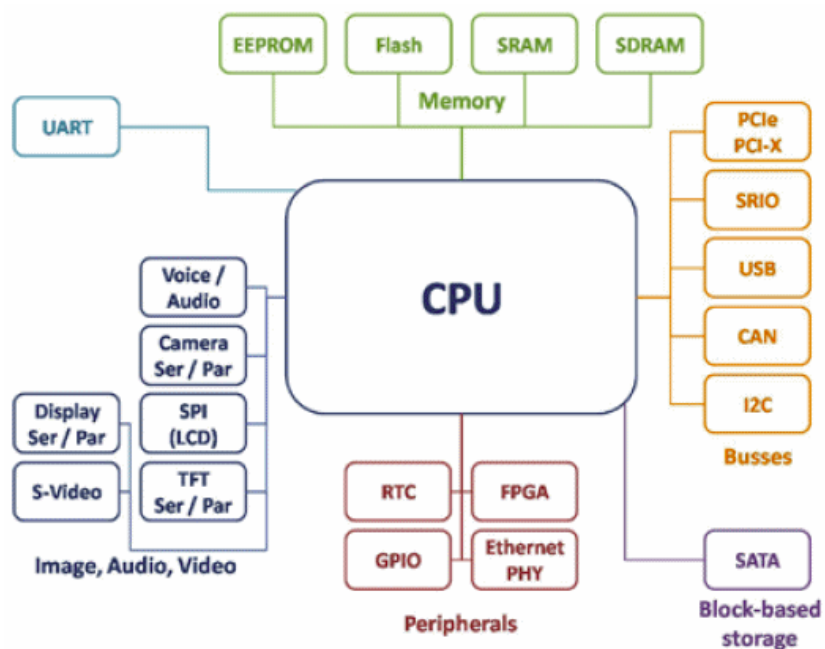
And yet another alternative is to first boot the system's operating system (Linux, for example), and then run a special validation application – or perhaps even run a real-world application.

If these techniques all sound a little "clunky" and extremely time consuming... that's because they are. What do you do when something goes wrong? Most of the time you sit around and scratch your head, break out one or more oscilloscopes and logic analyzers and connect them to various test points on the board, add some more tests to your code, and desperately try to work out what is going on. Surely there has to be a better way...

### *Hardware Validation TNG (The Next Generation)*

A modern system often comprises many different functions, including one or more central processing units (CPUs); audio and video capabilities; a variety of memory types (Flash, SRAM, SDRAM...); a number of peripheral functions, which may include one or more custom-designed Field Programmable Gate Arrays (FPGAs); various connectivity options (USB, CAN I2C...); and the list goes on.

Popular embedded processors like the OMAP™ 3 (based on the ARM® Cortex™-A8) and OMAP™ 4 (based on the ARM® Cortex™-A9) from Texas Instruments, or the PowerPC variants from the Apple–IBM–Motorola alliance, are extremely complicated. Because these processors are so popular, many IP functions are available for them, including interfaces to the latest-and-greatest peripherals. The end result is that validating such a system can be a nightmare (not the least that many teams continue to use the rudimentary hardware validation techniques that were briefly introduced in the previous topic).

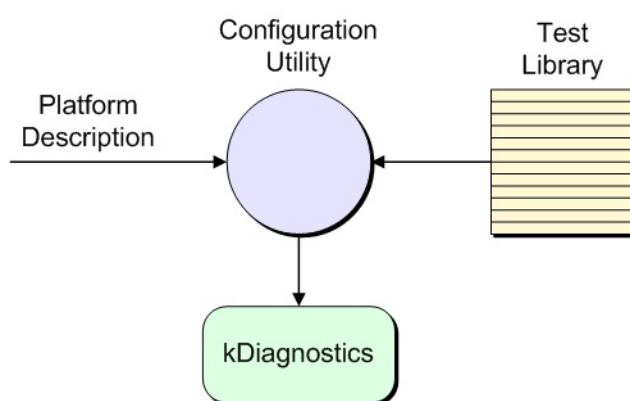


**Figure 2. A modern electronic system may comprise many different functions.**

The first issue is to determine who is in charge of generating validation software. In many cases this falls to the hardware development team, but they typically don't understand the software domain to the extent required to create validation tests that truly stress all aspects of the system. Alternatively, if the task is dumped on the software developers, they may not understand the underlying hardware to the extent required to develop a comprehensive suite of validation tests.

The engineers at Kozio ([www.kozio.com](http://www.kozio.com)) have come up with a solution. At the heart of this solution is an extensive parameterized library of pre-defined validation tests. This library covers the vast majority of standard system components used in OMAP or PowerPC-based systems. If your system includes a memory block of NOR Flash, for example, then the Kozio library already has a comprehensive suite of tests to validate this function; similarly for the other blocks comprising your system (additional tests can be quickly developed for non-standard components as discussed a little later in this paper).

All you have to do is provide Kozio with a schematic of your system in the form of a PDF file along with a register-map description of the various elements forming the system. The engineers at Kozio use this information to create a machine-readable description of your platform. This platform description is used to control a configuration tool that generates an executable called kDiagnostics® as illustrated in Figure 3.



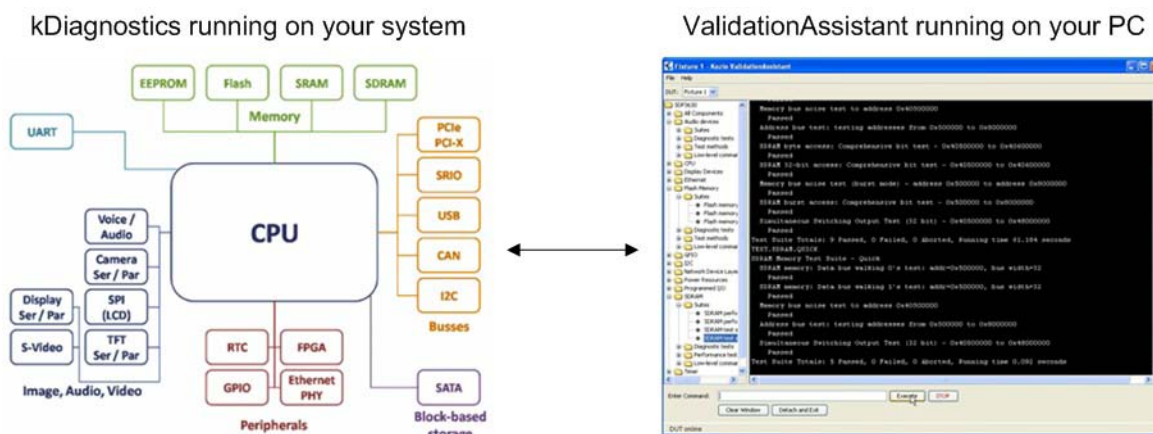
**Figure 3. Generating a unique validation environment for your system.**

In the case of any non-standard component like custom ASICs, SoCs, and FPGAs, the kDiagnostics solution automatically provides a memory map of the device, rudimentary read/write access of all registers, and basic programming capabilities (for commercial devices). If you provide the specifications of these devices to the engineers at Kozio, then they can create special tests for these components.

The kDiagnostics executable is unique to your system. It contains a test and diagnostics executive that runs on your target hardware and executes tests from a test library. This test library, personalized to your design, contains individual test methods and aggregated test suites, that together comprehensively validate the functional sub-systems of your custom hardware, including the CPU, all memories, peripherals, storage, connectivity, video displays, and audio channels.

Your kDiagnostics application can be auto-executed from any bootable device. If your system can boot from a USB port, for example, then your kDiagnostics executable can be loaded onto a USB memory stick.

Furthermore, the kDiagnostics application can be configured to run in different modes. For example, it can be configured to execute automatically on power-up and to provide a simple Pass/Fail result. Alternatively, kDiagnostics can be instructed to use one of your system's communications ports (Serial, USB, Ethernet...) to communicate with Koizio's ValidationAssistant™ application running on a PC as illustrated in Figure 4.



**Figure 4. Augmenting kDiagnostics with ValidationAssistant.**

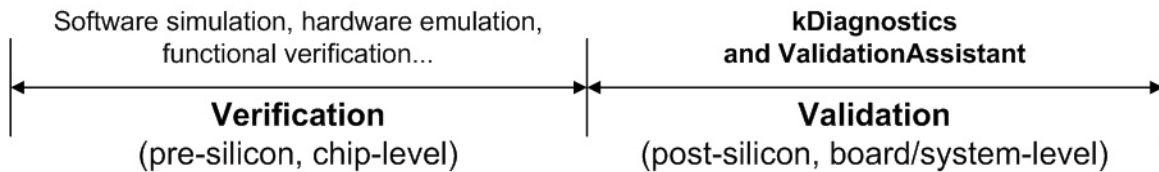
The ValidationAssistant option provides total visibility into the test and diagnostic process, maintaining hierarchical visual lists of test methods and suites, flexibly organized by component, test suite, low-level method, or test type (e.g. performance). In addition, users can configure methods and suites, initiate and control execution, and instantly see status reports on test completion.

### "It's validation Jim, but not as we know it..."

The Koizio solution is unique in that it can be used throughout all board/system-level phases of the development project. For example, it can be used for all levels of implementation, from silicon test boards, to development platforms, to the final deployed product. Furthermore, it can be used in multiple validation roles, including the following:

- In-System Diagnostics
- Diagnostics
- Embedded Functional Test
- Acceptance Testing
- At-Speed Functional Test
- System Test
- Manufacturing Test
- In-Field Power-On Self-Test

In addition to functional validation, the fact that users can interactively configure existing test methods, add new test methods, create unique test suites, and establish test loops, means that the Kozio solution can also be used to perform compliance testing (emission, power, performance...) and performance characterization.



**Figure 5. kDiagnostics and ValidationAssistant = Hardware Validation TNG**

In reality, Kozio's solution goes far beyond any form of traditional validation, because it is also of use for educating new teams as to the capabilities of a system. Consider a communications system whose hardware is designed by Company A, for example. It may be that this hardware is subsequently handed over to Company B for application software development and deployment. In this case, Company A may use kDiagnostics and ValidationAssistant to validate the hardware platform before handing it over. Meanwhile, the software developers in Company B can use kDiagnostics and ValidationAssistant to quickly ramp up their knowledge as to the new system's capabilities; for example, by using kDiagnostics to display working hardware settings and valid software initialization steps.

Using Kozio's solution means that software development engineers can be provided with a fully validated hardware platform upon which they can perform their final application installation and integration months earlier than today's traditional methods. Typical customers see savings of \$100,000 in test development and debug costs per project and they experience a minimum of three months reduction in time-to-market for each new product. The end result of using Kozio's solution is to increase productivity, reduce project risk, and reduce time-to-market (and time-to-profit).

So start using Kozio's solution and stop saying to yourself: *"Oh no, there's only 10 days to customer ship ... but it looks like we may have an XYZ problem!"*

## Bios

### Joe Skazinski

**Co-Founder and Chief Business Development Officer, Kozio, Inc.**

Mr. Skazinski co-founded Kozio in 2003 and operated as the Chief Executive Officer before becoming the Chief Business Development Officer. Prior to co-founding Kozio, Mr. Skazinski worked in director and managerial roles for storage solution providers, including Vicom Systems, Mylex, and IBM. While at Mylex and IBM, Mr. Skazinski was an integral part of several engineering teams that released five different storage controller products. His experience also includes the strategic business planning and execution of embedded software solutions, as well as selling to primarily every large company in the storage industry. He has more than twenty years of direct industry experience and is the co-author of over fifteen patents. Mr. Skazinski received a B.S. in Computer Science from Michigan Technological University.

### Clive “Max” Maxfield

**President, TechBites, Inc.**

Clive “Max” Maxfield received his B.Sc. in Control Engineering in 1980 from Sheffield Polytechnic (now Sheffield Hallam University) in England. Max began his career as a designer of central processing units for mainframe computers. Over the years, Max has designed all sorts of interesting "stuff" from silicon chips to circuit boards and brainwave amplifiers to Steampunk “Display-O-Meters”. He has also been involved at the forefront of Electronic Design Automation (EDA) for more than 20 years.

Max’s numerous technical articles have appeared in a wide variety of magazines, including *ED*, *EDN*, *Chip Design*, *EE Times*, *PCB Design*, and the electronics and computing hobbyist magazine *Everyday Practical Electronics (EPE)*. Also, Max he has held contributing editor or executive editor roles at *Programmable Logic DesignLine*, *Chip Design Magazine*, *SOC Central*, and *Everyday Practical electronics*. Max has also presented papers at technical conferences around the world [this year he will be presenting at the Embedded Systems Conference (ESC) Silicon Valley in April and ESC India in July].

Max is the author and co-author of a number of books, including *Designus Maximus Unleashed (Banned in Alabama)*, *Bebop To The Boolean Boogie (An Unconventional Guide to Electronics)*, *Bebop BYTES Back (An Unconventional Guide to Computers)*, *EDA: Where Electronics Begins*, *3D Graphics on Windows NT*, *The Design Warrior’s Guide to FPGAs*, *FPGAs: Instant Access*, and *How Computers Do Math*.

Max is the president of TechBites Inc. ([www.TechBites.com](http://www.TechBites.com)) – a unique mix of editorial and technical content with social networking – where he leads the Chip Design, FPGA Design, and PCB Design communities. He can be contacted at [max@techbites.com](mailto:max@techbites.com) or [www.techbites.com/Max-Maxfield](http://www.techbites.com/Max-Maxfield).

**About Kozio, Inc.**

Founded in 2003, Kozio provides packaged solutions for hardware validation and circuit board test to leading edge customers across the globe. Our tools are constructed from a library of over a million lines of proven diagnostic code that has been run thousands of times, on hundreds of designs throughout the whole product life cycle. Comprehensive hardware design validations are performed in minutes, not weeks, accelerating time-to-market with reduced risk and better allocation of engineering resources. Whether your application is in Engineering, Manufacturing, or In-Field test, Kozio is the trusted leader in In-System Diagnostics and your fastest path to reliable hardware. For more information, visit us at [www.kozio.com](http://www.kozio.com).