



## INTELLITECH: FPGAs AND SECURITY

25 October 2008: FPGAs and Security

INTERVIEWEE. CJ CLARK  
CEO, INTELLITECH CORP.  
TEL. 603-868-7116  
EMAIL. scansales@intellitechperiod.com  
COMPANY. INTELLITECH CORPORATION  
WEB. <http://www.intellitech.com/>

**Q. First of all, tell us a little bit about yourself and your responsibilities at Intellitech.**

A. I think most people in the industry know me from my work as chair of IEEE 1149.1/JTAG. My first job was back in 1978 for Plantronics/Wilcom, so I have seen a lot and I fortunately got involved in electronics at a very young age. I sit on two industrial advisory boards for the University of New Hampshire, I guest lecture on the IEEE series "Mission-Critical FPGA-based Embedded Systems" and I hold a few patents related to test and FPGA configuration. As CEO my responsibilities are mostly for setting the strategic direction for the company and anticipating our customer's needs rather than just reacting to them. I work with a great group of people at Intellitech, many for ten plus years now that do the rest of the work, executing on that strategic direction.

**Q. Everyone is familiar with Xilinx and Altera but not so much the vendors in the "FPGA ecosystem" more generally. Can you tell us briefly about your product offerings, especially as they relate to FPGA-based designs?**

A. Yes, thank you for asking. *SystemBIST* is our IC that provides key ecosystem functions for FPGA based PCBs. Those functions are FPGA parallel and serial configuration, FPGA Trojan protection, FPGA bitstream security, watch dog timers, periodic event control, power-on reset, embedded JTAG test and an in-the-field update engine. *SystemBIST* provides the FPGA configuration over both the high-speed parallel configuration bus as well as over JTAG. What's different than say using a commodity FLASH or PROM is that the sequence is programmable using our software GUI. You can specify how to program the FPGAs, for instance loading a different bitstream based on what size FPGA is present or based on what daughter boards is present. What we call 'conditional based configuration'. PCBs with FPGAs have power on reset ICs and watch dog timer ICs. What's different is that our built in POR and watch dogs are programmable in their behavior. At power-up a reset IC will just toggle reset, however with *SystemBIST* you can direct what you want to happen, perhaps hold the CPU reset low until after all FPGAs are configured and then release the reset. Similar things can be done with the watch dog, since it's integrated with our on-chip FPGA configuration engine and JTAG engine. Power-up sequences with the DC/DC converters can be controlled by *SystemBIST* and integrated with the FPGA configuration strategy rather than implementing with the mission mode CPU or home-brew CPLD design. We control the DC/DC converters via GPIO or program the voltage levels via the I2C interface. Since we have built-in JTAG based test, we can do things like voltage margining while running structural or at-speed based tests, all of course without handshaking with the mission mode CPU and firmware. *SystemBIST* is a smart board manager, it takes care of a lot of the housekeeping functions needed in FPGA based PCBs. It really can be a big mistake to integrate the infrastructure needed for FPGA programming, embedded test or in-the-field updates with the mission mode software. It adds to the complexity and needs to be developed entirely by engineers familiar with the software, CPU and firmware. *SystemBIST* keeps the non-mission functions, the 'auxiliary' functions; separate, which makes development, bring-up and

maintenance easier. There is less debug as the ecosystem strategy can be executed from our high level software tools and validated prior to embedding. With ad-hoc approaches, the tools aren't available, therefore debug and development is done manually by the firmware engineer. *SystemBIST* designs don't have to wait for the OS to boot for instance to do something as simple as run an embedded test or configure an FPGA. *SystemBIST* is easier for the FPGA engineer, who may not be experienced at writing embedded software, to use our GUI to develop the configuration strategy and the remote updating capabilities. Embedded test is easier as test engineers can import their manufacturing JTAG tests into *SystemBIST* without having embedded software engineering expertise or trouble the embedded firmware developer to do it.

**Q. Are your products competitive or complementary to those of the big FPGA vendors? In what ways?**

- A. They are complementary really. Any of my contacts within the FPGA vendors see selling the FPGAs themselves as their priorities, FPGA configuration is just a necessary thing that the FPGA companies provide application notes for. A key capability of *SystemBIST* is the embedded self-test, which is an area that FPGA companies just don't want to go into.

**Q. Security is a growing concern worldwide, and yet "security" can mean so many things. In the FPGA world, security is often explained as security against "reverse" product engineering. What other meanings of security do you see as relevant to FPGA-based designs?**

- A. That's a great question, I'm glad you asked. The emphasis on security has been against bitstream reverse engineering and copying by non-connected third parties. The current security, using AES encryption and a key programmed into the FPGA doesn't help too much against product overbuilding or cloning by connected parties. The keys are generated in the FPGA tools as plain text files which are typically sent to the contract manufacturer so they can be programmed in. Product overbuilding and cloning is typically done by knowledgeable insiders, ex-employees or unscrupulous contract manufacturer who builds more products to sell on the spares or other markets. That took place in the Cisco cloning case investigated by the FBI. PCBs need better protection, a PUF, a Physically Un-clone-able Feature. *SystemBIST* provides this with its unique on-chip customer ID and serial number.

Two other important areas are in security against hacking and security against Trojan bitstreams being loaded. If a key is present, depending on which FPGA you're using, the FPGA bitstreams that are not encrypted can be loaded into the FPGA. Battery backed security keys are easy to defeat by simply removing the battery or shorting the battery leads until the battery is dead. An attacker may not be looking to steal your FPGA design but to replace it with a trojan design. If you are making an ATM machine, voting machine, secure communications or gambling machine, think of how the operation would be compromised if an attacker could load in their own FPGA bitstream that mimics the operation but perhaps games the system in their favor or collects passwords. Any publicly available bitstream storage device or commodity flash is easy for a hacker to re-program using the same tools you used to program them with. Hackers might not be trying to load Trojans but get your hardware platform to do something else, maybe enable a feature that shouldn't be enabled.

**Q. Can you explain specifically what ways your product brings new levels of security? At a technical level, can you explain how it interacts with the on-board FPGA and thereby "secures" it?**

- A. Sure. The new security levels are in multiple areas. Our software generates a *SystemBIST* 'image' which is a binary representation of all of the things you would like your ecosystem to do, FPGA configuration, DC/DC converter control, embedded test etc. The image data is compressed and encrypted, it's not just straight bitstreams that can easily be discovered and replaced by a hacker. Further, they are tied to your customer code; even another customer who is using our tools will not be able to create an image that matches yours. *SystemBIST* has active security through tokens and secure hash. We provide a small amount of IP that you design into your FPGA with your 128-bit security key. *SystemBIST* sends challenge/response tokens to the FPGA over JTAG or over I2C to the FPGAs (one at a time) and expects a calculated hash response, which it compares internally to *SystemBIST*. If an FPGA fails to authenticate, then *SystemBIST* will execute a user defined operation, which could be something as simple as re-programming the FPGA or it could hold the entire PCB in reset until it is powered down. Similarly, the FPGA design is looking for this challenge/response pair and it can disable its internal logic if the data is not correct or never appears. This security becomes a bit more universal; it can be used with AES capable FPGAs or FPGAs without AES. Copying the bitstream will not help an attacker, as the operation won't work in another system. Loading a trojan bitstream with JTAG won't work either as *SystemBIST* is constantly checking for the FPGA authenticity. There is no non-volatile storage that the unauthorized bitstreams can be stored in. Cloners can't buy *SystemBIST* devices with your customer code and security key or the software to generate new images, so it becomes impossible to make more copies of your product than you authorize.

Our update mechanism is also secure. It is far superior compared to sending bitstreams over the internet or open text files like STAPL. STAPL is tough to use these days anyway, it's JTAG only, doesn't support parallel FPGA configuration needed for speed and is an open text file that is very easy to hack. *SystemBIST* checks for authenticity of the FPGA update, the update is correctly targeted for this system, it's the right version and it has the proper secure hash algorithm values in it. Bits that get modified on the way to the product by an attacker will cause the hash to be incorrect and prevent *SystemBIST* from performing the update.

**Q. What's the connection between JTAG test and FPGA configuration? And why is embedded test important, how does it differ from software based self-test?**

- A. That's a great question as well. Software based functional test is typically developed by someone who is expert with the system architecture and has the ability to write embedded firmware. IEEE 1149.1/JTAG in contrast was developed so testing could be performed without being experts in the mission mode logic. Home-brew functional test is not well understood by third parties like contract manufacturers, they can't change the test or diagnose failing boards very well, and it's not cost effective for them to become experts in the functionality of one customer's product. JTAG/1149.1 testing however is very well known by the CM. JTAG tests most likely are already available for production, with *SystemBIST* it becomes quite easy to import and re-use them for embedded test in the field. There is much more automation in test generation with 1149.1 compared to software functional test which is typically a manual effort. 1149.1 tests tend to be much better at pinpointing a fault than a functional test.

FPGAs typically have to be configured in order to maximize test coverage. Consider an FPGA with LVDS I/O connected to an LVDS device. An unprogrammed FPGA's I/O is LVTTTL, so the 100-ohm termination resistor will look like a dead short to any JTAG pin toggling. The FPGA would have to be configured to run an interconnect test across the I/O to the other device. In today's systems, we program the FPGA's multiple times, we download test helper circuits (which we call TEST-IP) into the FPGAs to facilitate at-speed testing. We download BER engines to the FPGAs to test the SERDES connections at speed. We download memory BIST engines to the FPGAs to test the DDR2/DDR3 connections at speed. With *SystemBIST*, the user can embed these tests and use the built-in periodic suite to get asynchronous voltage margining for instance during the DDR or BER tests. You could try to do this all with firmware and the mission mode CPU, but it's a bit more messy. Functional test usually is an end-to-end test, where for instance

tests of multiple SERDES connections are tested together, with a dependency on an error-free DDR memory interface to store the packets. The PLL and DC-to-DC converter have to be functioning as well. With JTAG based BER tests checking for proper DC levels and checking that the PLL jitter is not out of spec is part of the start-up test before the channel is even at-speed. With whole board functional test, if one item is wrong it's hard to tell what went wrong since so much of the system is functioning and interacting. With directed tests like BER, we concentrate on testing one element. JTAG based at-speed testing or what we call 'emulation based testing' is more directed, its essentially datasheet testing with little emphasis on the top level functionality of the systems.

In our online presentations we show this layered approach to embedded test, a pyramid representing the entire test development. The manufacturing tests which already exist can be imported, this provides us with a base-line structural test for the PCB, checking all the interconnects, checking connections to mezzanine cards and DDR memories. The next level is the at-speed testing, this is where the FPGAs are programmed with special designs which can be executed via JTAG, then the mission mode functional test can be added at the top. With *SystemBIST* you have a lot more known variables and a lot less work to do. CPU based functional test can have grey areas where it's not known if a failure in the field is due to the software or to the hardware. *SystemBIST* test gives you that extra data point; it tells you if the hardware is failing with no ambiguity. *SystemBIST* stores all of the failures in non-volatile memory, for retrieval by the mission mode CPU or manually with a JTAG tool to extract the failures. There is never the possibility of a NFF, no fault found as the failures are always stored and can be used with our software to determine a net/pin or register level fault.

There is a new effort within the IEEE called P1687 or IJTAG. This standard uses 1149.1 to access on-chip instruments like our FPGA BER and DDR BIST, however these instruments will be on Asics and commodity parts. In my opinion, the momentum is building towards more structured test via 1149.1 and away from software based functional test. It is much like the history of integrated circuit test. IC test was done with functional testers years ago, but as the logic complexity increased it became too expensive to develop manual functional tests. Structured tests via scan were done since ATPG – automatic test pattern generation could be done for stuck at faults. Then tools became more robust and could generate transition faults and path delay faults via scan; now we have logic *BIST* and on-chip memory *BIST*, all which run at speed. But very little of the test is based on the functional design of the IC.

**Q. How much does *SystemBIST* cost? Would you explain your business model and any different options in terms of engagement?**

A. *SystemBIST* is roughly \$10-15 depending on your volume. That cost of course is completely recouped in removing other FPGA configuration methods, reducing the FLASH memory size, removing CPLD power sequencers, power on reset chips, watch dog timers and other parts. Our business model is that we sell the silicon, we license the software tools that go with it, and we provide services that guarantee our customers success. Once your design team is using a structured method for FPGA configuration and embedded test, where the infrastructure is not mixed in with the mission mode functionality, then engineering functions that typically are done in-house can be outsourced to third parties, such as Intellitech, who can work simultaneously on the ecosystem without being in the critical path of the mission mode firmware development.

**Q. How would you recommend a potential customer find out about *SystemBIST*? Do you offer online webinars? Training seminars? Demos? How can one “try” before “buying?”**

A. Much of this is on our website. You can see a webinar, take a look at the reference designs and read our white papers on *SystemBIST* on the website. We have a new live webinar coming in

December. There is an evaluation PCB as well. Our application engineers will help the designer get *SystemBIST* designed into their PCB.

**Q. Thank you for this interview.**