Surviving State Disruptions Caused by Test: the "Lobotomy Problem"

By
Kenneth P. Parker
Agilent Technologies
Loveland, Colorado

kenneth_parker@agilent dot com

Surviving State Disruptions Caused by Test: the "Lobotomy Problem"

Kenneth P. Parker
Agilent Technologies
Loveland, Colorado

kenneth_parker *at* agilent *dot* com

Abstract

The practice of initializing a board or system for testing purposes is not an exact science, but rather, pursued empirically and with little help from IC designers. This paper examines some of the issues and trends that justify adding features to IEEE 1149.1 that will facilitate safe, fast and effective initialization of a board or system, to get it ready for testing and to leave it in a safe state upon completion of testing.

1 Disclaimer

There is a new working group for IEEE 1149.1 that is contemplating a revision to the standard which could implement some of the ideas presented in this paper (see [IEEEWG]). I am a member of that group, but the content of this paper may not reflect their final thinking and results, if any. Opinions stated throughout this paper are my own.

2 Introduction

Imagine for a minute that you are a printed circuit board. You have 240 silicon devices, and over 1,000 analog devices ranging from DC-to-DC converter circuits, to termination resistors, to high-frequency bypass capacitors. Some of your ICs are wildly complex systems in their own right. Other components consist of empty connectors that would connect some of your nodes to other subsystems such as PCIe I/O cards or extra memory. You are the flower of high technology, hot out of the reflow oven and ready to go – but where? To the test area, of course!

You are placed on an In-Circuit Test fixture and find yourself pressed down on a "bed of nails". (Ouch!!) But now, you feel the life-giving flow of current in your circuits as your power nodes are connected to voltage sources. Here comes the 40 volt supply, and this lights up your DC-to-DC converters, but, not all at once. First the 5 volt section turns on, and it gradually stabilizes. Then, after a few hundred milliseconds, on comes the 3.3 volt supply. Pretty soon, your ICs begin to wake up. They have been carefully choreographed to turn on in a certain order, following a power-up sequence determined by your design team. Your main processor begins to fetch instructions from the base address of your boot ROM. It sends commands to other neighboring ICs to get them going on their appointed tasks. The DC-to-DC converters adjust to changing current loads. Things are really getting interesting now. Heat is beginning to build up in some of your devices as they begin to hum along nicely. Life is good and getting better.

Suddenly, with absolutely no warning at all, 98 of your ICs seem to go berserk. They have stopped interacting in a rational way. They seem to be completely mindless of their mission. They ignore commands and, worse, they stop sending meaningful responses back. They ignore the master clock. They seem to be doing something, but are very slow now. The DC-to-DC converters are trying to respond and are having trouble finding equilibrium. Their voltages are not as stable as you'd like. What the heck has happened here?

Well, this situation continues for several seconds. You notice that some activity is occurring on a small number of nodes that are receiving foreign signals from the bed of nails. This might be the cause of problem, but, before you know it, they stop. The ICs are not doing much at all now, just like they've lost their minds. Several are in a tight loop sending messages but ignoring the answers. Others are completely silent. But now there's a worrisome development – the main processor is trying to execute code from a bogus address – who knows what that code might do! And the signals it is now sending to the DC-to-DC subsystem are complete nonsense. Of course, the voltages are now changing again and this is also a troubling event. Suddenly, the main 40 volt supply goes to zero without any warning. What a day this has been!

* * * *

What this thought experiment has attempted to motivate is what a board might experience when, after being powered up, it is tested with 1149.1 (or 1149.6) Boundary-Scan [IEEE01], [IEEE03]. This rather disrupt-tive event interrupts the board's normal activities by switching a large number of IC I/O pins from "normal" mode to "test" mode. This event is completely unsynchronized with any current activity, so that the internal logic of each IC may suddenly see completely illogical states, clocks may be disconnected, the outputs suddenly enabled when they were tri-stated, et cetera. What is the effect of this on the board? Well, for all the nodes that are contained by Boundary-Scan device pins, we know that they are participating in some form of carefully crafted test. The other nodes are probably responding to these signals, but it may not be clear how. Further, when the Boundary-Scan tests complete, the pins states are returned from test mode back to what the 1149.1 standard euphemistically calls "normal" mode again. But few would expect that normalcy will ensue. Indeed, it should be a great worry that undefined, possibly harmful results could follow before you could get the power turned off. It's a bit like an empty car careening down a road with a brick on its accelerator.

This situation was termed the "Lobotomy Problem" in [Park03] (in 1992, first edition), as an anthropomorphic allusion to surgical disconnection of the pre-frontal lobes of the brain from the main cortex – a frontal lobotomy. (This controversial medical procedure was developed in the mid-1930s and was largely replaced with drug therapies by the 1960s. Who can forget Jack Nicholson's portrayal of a lobotomized inmate in "One Flew Over the Cuckoo's Nest" in 1975?) We regularly lobotomize millions of complex logic boards and systems every year without asking if there might be a less randomly violent way to treat these potentially delicate entities.

I have found no "smoking gun" where collateral damage due to Boundary-Scan testing has been documented, but I think that is due both to luck and a lack of root-cause analysis. The random nature of when lobotomization occurs in a board's otherwise natural course of events may well hide the occasional harmful effects as a part of "normal" yield loss, i.e., we won't find what we don't look for. And there is the possibility that damage does not occur, but lifetime degradation is a result.

The rest of this paper will discuss technology trends that will make this problem of increasing concern, and examine a potential solution toolset proposed as an addition to the 1149.1 standard.

3 Technology Trends

There have been perhaps 12 to 14 cycles of Moore's Law in the last 20 years, which is the time since the IEEE 1149.1 standard came into being. Back then the ascendant large devices were typified by the 68040 or the 80486. Not surprisingly, there have been some modest technological changes since then. Some of these are:

- Programmability – devices can be configured for functionality after they are manufactured and within their application, including the voltage levels they produce and accept on their I/O pins.
- Clocking – devices once accepted a direct clock signal; now they take that clock signal and multiply it up with a phase-lock loop (PLL) to a much higher rate inside the IC. Indeed there are often multiple independent clocking domains within an IC.
- Power – devices consume much more power with more transistors leaking more current. It is quite common for devices to have several supply voltages. Devices get hotter, faster. The necessary heat sinks can be massive, bolt-on assemblies that are incompatible with test fixtures.
- Power management – devices are designed for power and thermal efficiency. Thus they can "turn off" subsystems which are idle or at thermal risk.
- I/O interfaces – these are vastly more complex, with serial protocols, embedded clocking, very high data rates and analog "tuning" for ultimate performance.
- Complex bootup sequences – these incredibly complex devices have a lot of work to do just "waking up".

Boards that use these ICs have also evolved. They are (of course) far more complex. Since the devices on these boards use several lower voltages but much higher currents, it has become common for a board to regulate several lower voltages from one much higher voltage by use of on-board DC-to-DC converters, often implemented with discrete analog components.

It is quite common for the act of applying power to a board to be a complex, sequential series of actions taking hundreds of milliseconds or more. Disturbances in the regulation process can produce "exothermic events" very quickly, sometimes with spectacular (also expensive) effect.

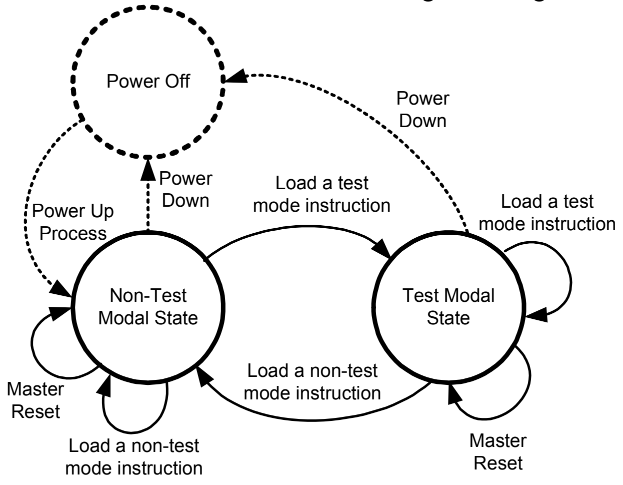
The heat generated in these devices is also problematic, so exotic cooling mechanisms are more common these days. That said, these cooling structures are often not in place during board testing for practical reasons. This can place restrictions on how long boards are powered during testing.

Circuit access used by bed-of-nail testers is (as usual) more difficult to achieve, placing more limits on what can be tested without the assistance of Boundary-Scan-based tests. Thus, we see increasing numbers of ICs on boards that contain 1149.x technologies. This same lack of access means we cannot detect as many potential shorts on boards while the power is still off. Testing for shorts with the power on is more risky, but also inevitable.

Today's board test or system test engineer is finding increasing challenges in guaranteeing the safety of boards (and fixtures and operating personnel) due to many of these trends. At the same time they are under increasing pressure to decrease test costs as well. It can be very difficult to figure out solutions, especially when a board's design team has not given the test engineer some "hooks" to use in solving these problems.

4 Current 1149.1 Modal States

The IEEE 1149.1 standard views silicon devices as having two basic operations, here called "modal states". The first is "non-test modal state" (also referred to as "mission mode" in the literature) and the second is "test modal state". A device will toggle between these two modal states when a "test mode instruction" such as EXTEST is loaded versus a "non-test mode instruction" such as BYPASS is loaded. See the diagram in Figure 1.



Power Off
Power Down
Power Up Process
Power Down
Load a test mode instruction
Load a test mode instruction
Non-Test Modal State
Test Modal State
Load a non-test mode instruction
Master Reset
Load a non-test mode instruction
Master Reset

Figure 1: The basic modal state model for 1149.1.

This model also shows (in dashed lines) that there is another state of being powered off and that you progress from the power off state to the normal operation mode via some sort of power up or "boot" process. In some devices this is just by asserting the TRST* pin if it exists. In others, it is accomplished as a natural result of powering up the device as required by rule 6.2.1a) of the standard [IEEE01]. There is also a synchronizing sequence (5 TCKs with TMS held high) that can take the device to the Test-Logic-Reset state, but this is not required as a part of power up. Note the synchronizing sequence has the effect of jam-loading a non-test instruction, so it takes a device from test operation to non-test in such case.

The "Master Reset" transition (think "reboot") is accessible in the non-test modal state, but in the test modal state, that signal is ignored and does not disturb testing activities.

The transitions in question here are those from "non-test modal state" to "test modal state", and vice versa. Both are triggered by loading instructions and so occur on the falling edge of TCK in the Update-IR TAP controller state, which is not synchronized to particular system activities. For example, a processor may be about to receive an instruction it is fetching from memory, but just as this instruction opcode arrives, the processor switches to EXTEST (assume the Boundary register cells are control-and-observe cells) so the opcode is blocked. How does the processor react internally? Also, the processor outputs are suddenly disconnected from their surrounding devices. What happens to them? This is the genesis of the lobotomy problem. Once the system is so violently disturbed, it is quite difficult to even imagine what the result could be, but it is easy to imagine the worst.

Similarly, what happens when testing is complete and "non-test modal state" is re-entered? Again, we can only imagine (and cringe). Notice too that whenever the TAP enters the Test-Logic-Reset (TLR) state, this also forces the same transition. Since many standalone Boundary-Scan tests begin in the TLR state and end there too, this virtually guarantees lobotomization.

5 The "Real" Modal State Model

The reality of 1149.1 modal state changes is captured in Figure 2, where the lobotomized modal state is a "trap" that prevents the system from getting back to normal. The only path back to normal is by cycling power or possibly by the Master Reset process (in **red**), if that process is still workable in the lobotomized state.

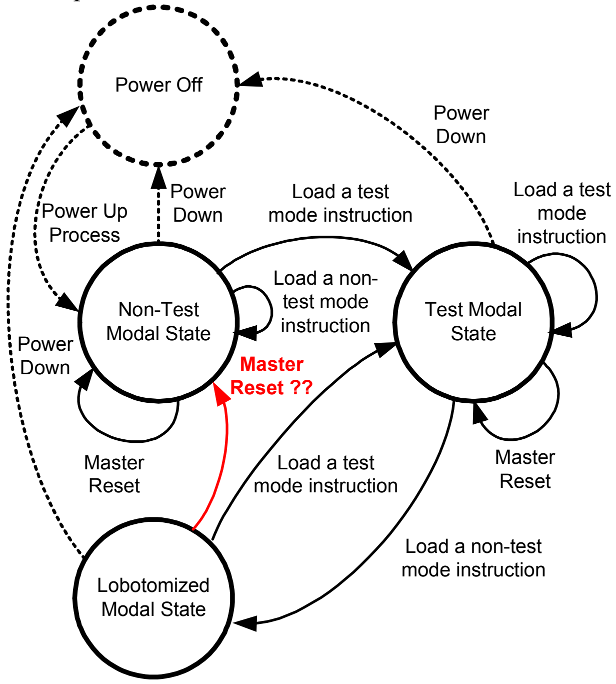


Figure 2: Basic model including lobotimization.

Note that the modal state concept in Figure 2 describes individual ICs, or full systems containing at least one 1149.1 IC.

The next section describes a way of dealing with the lobotomy problem by the assignment of a new modal state. This modal state would be accessed via new 1149.1 instructions.

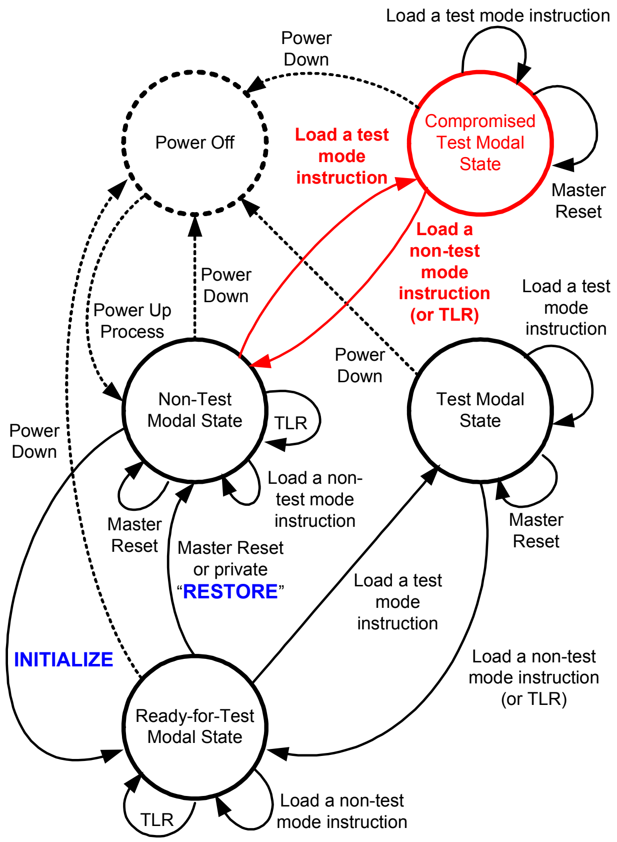
6 The "Ready-for-Test" Modal State

As IC technology has evolved, there are a lot of internal features inside ICs that take some time to become ready for normal operations to commence. For example, internal phase-lock loops have to acquire a frequency and lock onto it. Internal charge pumps may have to create internal voltage references. High-speed I/O paths may have to be "tuned" for optimal performance. Selectable I/O pin voltages may need to be programmed, and so forth. These internal "boot up" processes may be application dependent, so the board-level context will affect the end results. Once these ICs are booted, they then may access system boot code and begin higher level activities like loading operating system code, accessing storage devices, etc.

In the middle of all of this, a manufacturing test could suddenly start up, lobotomizing the system. However, by adding a new concept of a modal state dedicated to the test problem, it is possible to avoid some of the nasty implications of undefined behaviors. This modal state can be called "Ready-for-Test", and it is inserted between the non-test modal state and the test modal states. See the diagram in Figure 3.

Upon powering up, a device will enter the non-test modal state as always. There it will stay until power down, or, a new "**INITIALIZE**" process is performed via new 1149.1 instructions. (For now, please ignore the **red** transitions in Figure 3.) The **INITIALIZE** process will take a period of time and/or some number of TCK and/or system clocks to produce a state that is compatible for testing. Some of the things **INITIALIZE** might do are:

- Shut down phase-lock loops, or switch them to a benign internal reference oscillator, before EXTEST disconnects the I/O.
- Put power-hungry subsystems that can be powered down into low power modes. This is particularly important when heat sinks or fans may be missing on devices, which is common during board manufacturing test.
- Put internal state machines and analog subsystems into quiescent modes. This reduces power consumption and noise generation.
- Disable internal busses to remove the opportunity for bus contentions.
- Turn on 1149.x circuitry needed for testing. For example, IC designers implementing IEEE 1149.6 [IEEE03] have complained that the test receiver circuits are power hogs that they only want to turn on when needed for testing.
- Configure I/O properties of the device's pins so as to be compatible for testing. For example, voltage levels for drivers and receivers can be selected and drive strengths and slew rates can be optimized for the rather pedestrian needs of Boundary-Scan testing.



Load a test mode instruction
Power Down
Power Off
Load a test mode instruction
Compromised Test Modal State
Master Reset
Load a non-test mode instruction (or TLR)
Power Down
Power Up Process
Load a test mode instruction
Power Down
Non-Test Modal State
TLR
Test Modal State
Power Down
Load a non-test mode instruction
Master Reset
Master Reset
Master Reset or private "RESTORE"
Load a test mode instruction
INITIALIZE
Load a non-test mode instruction (or TLR)
Ready-for-Test Modal State
TLR
Load a non-test mode instruction

Figure 3: Adding a new "ready-for-test" modal state.

This last example implies that some application specific configuration data may need to be supplied and indeed this is the case. Further, it is anticipated that more complex devices will need **INITIALIZE**, but simpler devices may not. Finally, it would be best if any devices in a chain of devices that need **INITIALIZE** will be able to execute such routines in parallel while the others "stand by". When the **INITIALIZE** process is complete, then the device is "Read-for-Test", meaning test mode instructions like EXTEST will be effective and the devices will be well-behaved during testing. It's like announcing to them: "We are going to start testing soon. Do whatever you need to prepare."

The **INITIALIZE** process is thus invasive to the "normal" (non-test) operation of the device (and system). The process systematically calms the system into a "safe and cool" state where subsequent testing can commence without fear of nasty side effects. However, it has the important characteristic of being able to respond to a "Master Reset" by returning the device back to the non-test modal state. It is also envisioned that an optional, perhaps proprietary "**RESTORE**" instruction could be available to invoke this same master reset process via the TAP.

Another important property is that from the ready-to-test modal state we can proceed back and forth to the test modal state, and the conditions set up by **INITIALIZE** remain in effect. Thus, a sequence of Boundary-Scan tests that need this calm, ready-to-test state can each start and end in the Test-Logic-Reset state without need of executing the **INITIALIZE** process between each. Since the time (and data) needed to **INITIALIZE** all such devices on a board could be significant, it is more efficient for the state created by **INITIALIZE** to be persistent.

The transitions marked in **red** in Figure 3 show the case where test mode is entered without benefit of the **INITIALIZE** process. This takes the device (or system) to test mode, but the necessary conditions to support testing may not be in place. This is termed a "compromised test modal state". Testing this way may or may not be successful, particularly if I/O setups needed for test are not the default states of the I/O pins in devices that utilize **INITIALIZE**. These **red** transitions should thus be avoided.[1]

The next section describes some new, optional 1149.1 instructions that support some of these concepts. Not all are currently supported (as will be noted) since there is some continuing debate about which concepts are practical or impractical to support.

7 New 1149.1 Initialization Instructions

The 1149.1 Working Group is studying the addition of two new instructions to support a formal concept of initialization for testing. Both are optional, so devices supporting an initialize process will implement at least one of them.

7.1 The INIT_SETUP instruction

This optional instruction is non-intrusive (like PRELOAD or BYPASS) and is used to set up application-dependent data needed to support 1149.1 testing. It targets an INIT_DATA data register which will receive parameters needed to set up the environment needed by test instructions.

The classic use for this instruction is to set up I/O pin operating parameters. For example, many devices have programmable logic levels on their I/O pins, allowing them to be used in multiple environments, such as with 3.0 volt logic, or 1.5 volt logic. The drive voltage levels and receiver thresholds on such a device will need to be configured for the device's environment. Thus multiple instances of the same IC on a board may end up with different configuration loads into their INIT_DATA registers. The configuration data does not immediately reprogram the I/O behavior (as this would be invasive to IC functionality), but when a pin-permission instruction like EXTEST (or INIT_RUN described next) is loaded, putting the device into test mode, the I/Os respond to the configuration data held in INIT_DATA at that time.

Other setup data may be needed as determined by the device designer. Thus the INIT_DATA register may be logically divided into fields that serve separate purposes. The working group is developing new BSDL (Boundary-Scan Description Language) constructs for conveniently labeling and describing these fields. A test engineer developing a test program for a board or system will need to set up a "side file" of configuration data for each IC that supports the initialization process. A user interface that reads the device BSDL will be able to present the options to a test engineer with meaningful labels, so s/he can make the right choices for loading INIT_DATA of a given device. These choices will be determined by the device's environment, and information from the device's data sheet, if needed.

7.2 The INIT_RUN instruction

This optional, intrusive instruction drives the internal state machines needed to execute the initialize process, which may use data previously provided by INIT_SETUP. The INIT_RUN instruction is similar to the 1149.1 CLAMP instruction in that it uses data in the Boundary register to fix the output drivers at a predetermined state for the duration of operation of INIT_RUN. This means a preceding PRELOAD sequence is needed to set up the Boundary register with data to define these output states. Also, if the device has programmable I/O states, these should be set up in advance by INIT_SETUP. More detail about this is given in section 8 below.

INIT_RUN differs from CLAMP in that it targets an INIT_STATUS register between TDI and TDO (rather than the Bypass register). INIT_RUN also has a time specification for its duration, which may be some number of TCK cycles, system clock cycles, or, elapsed time. The INIT_RUN instruction may take an appreciable time to execute, and while this is occurring, the I/Os are clamped to a consistent state. The status register will indicate a minimum of a "Done" bit and possibly a "Success" bit too. (Success indication is a matter of some debate among IC designers.)

The Done bit is expected to be set by the time the

[1] Changes to BSDL are needed to support INITIALIZE. A device that supports INITIALIZE would have a new BSDL Component_Conformance statement that would not be recognized by legacy software. Such software would not be able to process the new BSDL and thus force the question of upgrading it.

duration specification has been met, but it could be set earlier if the device designer has options that can complete earlier. Thus, polling could be performed to end the initialize process earlier. The duration is a "maximum" specification.

The Success bit (if provided) indicates that the INIT_RUN process's self-assessment is that it completed its task successfully, or sensed some sort of internal exception. In the event of a status failure, it may be pointless to continue the test. Additional INIT_STATUS bits can provide more information as to what is being sensed, to aid in assessing what should be done with the board.

7.3 A proprietary "RESTORE" instruction

A RESTORE instruction is not currently being considered by the working group. This optional, proprietary instruction provides a TAP-accessible means of re-booting a device, and provides the same effect as asserting a Master Reset signal. It is considered proprietary based on the belief that the 1149.1 standard should not provide a body of rules about how this is implemented beyond providing a name and invocation method for such a function. Because activating such an instruction is a conscious act, it should work in either non-test or test mode, whereas, triggering a Master Reset signal can only work in non-test mode.

7.4 Persistent "Ready-for-Test" modality

The working group has considered the importance of the "Ready-for-Test" modal state being persistent even when the TAP travels into the Test Logic Reset state. In section 6 it was argued that this could allow a single invocation of INIT_SETUP/INIT_RUN to prepare a Boundary-Scan-based board or system for a group of independent Boundary-Scan tests that were successively executed. However, IC designers have indicated this could be difficult or expensive to achieve in some device architectures. Thus, the Ready-for-Test modal state is (currently) not envisioned to persist across a TLR boundary. This means every independent Boundary-Scan test must, as its preamble, perform INIT_SETUP/-INIT_RUN before proceeding into test functionality.

The non-persistence of Ready-for-Test changes the modal state diagram to that shown in Figure 4. Here we see the "Lobotomized" modal state reappearing as a destination from active testing. The initialize process can restore the Ready-for-Test modal state from the lobotomized state, or a Master Reset (or RESTORE) can return to the non-test modal state. The TAP-based RESTORE instruction can also "abort" a test and move back to the non-test modal state, which is something a Master Reset cannot do from the Test modal state.

A set of manufacturing tests for a board would commence as follows:

1. Turn on power, board moves to Non-Test modal state and boots up. The bootup process may or may not complete before the test starts.
2. Use PRELOAD to set up first test events. (Stay in Non-Test modal state.)
3. Perform the INITIALIZE process in all ICs that support it (see detail in section 8). This moves to the Ready-for-Test modal state.
4. Proceed into testing with EXTEST in all ICs; now in Test modal state.
5. End test, TLR causes Lobotomized modal state to be entered. Quickly proceed to step 6.
6. Subsequent tests progress from Lobotomized modal state via step 2 above. Then power down or cause a Master Reset (or RESTORE) to bring back Non-Test modal state.

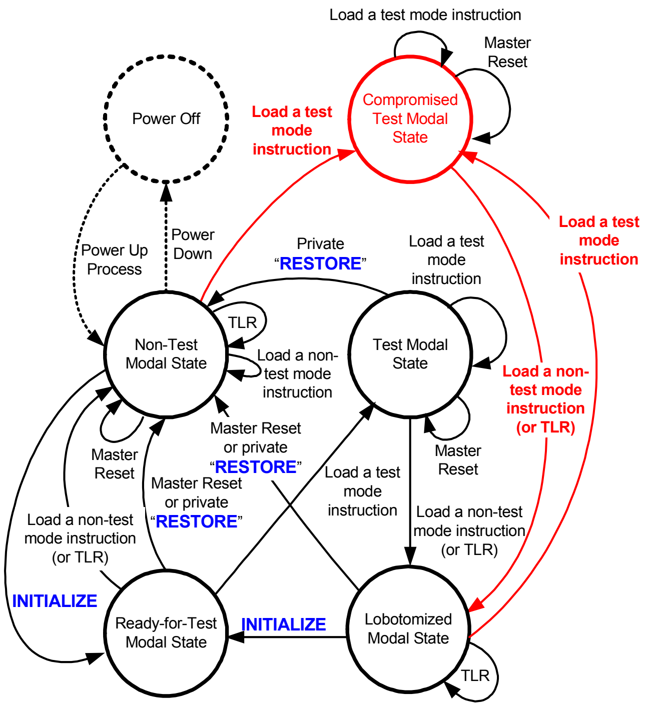

Load a test mode instruction
Master Reset
Compromised Test Modal State
Load a test mode instruction
Power Off
Load a test mode instruction
Power Up Process
Power Down
Private "RESTORE"
Load a test mode instruction
Non-Test Modal State
TLR
Load a non-test mode instruction
Test Modal State
Load a non-test mode instruction (or TLR)
Master Reset or private "RESTORE"
Master Reset
Master Reset or private "RESTORE"
Load a test mode instruction
Master Reset
Load a non-test mode instruction (or TLR)
Load a non-test mode instruction (or TLR)
INITIALIZE
Ready-for-Test Modal State
INITIALIZE
Lobotomized Modal State
TLR

Figure 4: Modal states for non-persistence of Ready-for-Test (note some power-down transitions omitted for clarity).

If one of the series of test should fail, then there is a question of whether to proceed with the following tests or quit testing. Quitting may be the preferred action if a test has discovered shorts on the board; shorts could cause collateral damage. If quitting is the required action, then powering down the board may be performed next.

8 INITIALIZE in Boundary-Scan Testing

Before examining the implications of initialization on Boundary-Scan tests, it helps to map out a typical Boundary-Scan test that would check for interconnect defects on a board with two garden-variety Boundary-Scan ICs. The basic process is shown in Table 1 (at the end of this paper). It shows 8 steps, the first 3 of which are performed in Non-Test mode. These are used to set up the first test vector behind the device drivers. At step 4, the EXTEST instruction becomes operational which both lobotomizes the devices (and board) and causes the first test pattern to propagate out of the Boundary-Scan-controlled drivers across the board to Boundary-Scan receivers. Steps 5 and 6 iterate this until all test patterns have been propagated. Step 7 flushes out the captured results of the last test pattern, while filling the Boundary registers of all the devices with a benign "safe" pattern. Finally, at step 8, the Test-Logic-Reset state is entered, completing the formal Boundary-Scan test. This also returns control of all Boundary-Scan drivers back to the system logic, whatever it may be doing.

Now look at what the INITIALIZE process adds to the test, as shown in Table 2 (at the end of this paper). This example considers a chain of 4 devices. Device1 has no support for INITIALIZE at all. It could be a legacy component, or, a simple device that needs no initialization. The second device, Device2, implements INIT_SETUP only. It may have some I/O pins that need to have their voltage levels set accordingly so they can communicate with surrounding circuitry. Device3 implements INIT_RUN only. It may have some internal power domains that can be shut down to reduce heat dissipation during testing. Finally, Device4 implements both INIT_SETUP and INIT_RUN. It has I/O levels to set, and other internal functions that need to be managed before test begin. Both Device3 and Device4 produce a "Done" bit in their status registers. (In this example, we do not consider the import of a "Success" bit and what the test would have to do if Done was set, but Success was not set.)

Table 2 is an expansion of Table 1, with 5 new steps inserted between steps 3 and 4 of the original test (highlighted in yellow). Essentially, steps 1-3 establish the first test pattern in the Boundary registers of the devices, but before switching to EXTEST to begin the test, the INITIALIZE process commences. Then step Init1 loads either BYPASS or INIT_SETUP in the devices. Both are non-test instructions so the devices continue doing their functional operations. At step Init2 the two devices that support INIT_SETUP are then loaded with setup data, while the other two continue with BYPASS. If none of the devices in a chain support INIT_SETUP, then steps Init1 and Init2 could be skipped.

Next, at step Init3, all 4 devices switch control of their drivers to the Boundary register, executing either EXTEST, or INIT_RUN. This lobotomizes the devices (and the board). Back at Step 3, the data needed for the first test pattern was preloaded into the Boundary registers. So at this point in time, that pattern is being propagated across the board, but, there is more initialize work to be done yet.

At step Init4, devices 3 and 4 are shifting out their status registers. But, devices 1 and 2 are doing EXTEST, so we must shift the same data from Step 3 back into them so that their drivers will continue to hold the first pattern data constant. This has the effect of making status checking fairly data intensive, if the Boundary registers in devices 1 and 2 are long. At Step Init5, the "Done" bits shifted out are checked to see if INIT_RUN has completed. If not we go back to Step Init4 and try again.

Once step Init5 finds both devices 3 and 4 are done with INIT_RUN, our original test process commences at Step 4, where all the devices are set up to run EXTEST. The Boundary register data set up by Step 3 is still in place, so the "standard" Boundary-Scan test proceeds as before. The major difference is that the time between the start of the first test pattern and the second test pattern is lengthened by the time required to do the initialization.

9 Conclusion

The "Lobotomy Problem" has been described, with motivation for why it should no longer be ignored. A well-defined and controlled process for preparing for test mode is proposed. Discussions with IC designers and test engineers as well as board and system architects have been solicited. Their input is reflected in a proposal for 1149.1-based support of an **INITIALIZE** process.

However, perceived silicon design difficulties have precluded the creation of a persistent "Ready-for-Test" modal state, resulting in Boundary-Scan tests that still leave devices (and boards) in lobotomized states when testing completes. Test engineers will still need to evaluate what this means and how they should deal with it.

10 Acknowledgement

I would like to thank the 1149.1 INITIALIZE Tiger Team that has been studying the INIT question for a number of months. In particular, Carol Pyron and Carl Barnhart have provided leadership, guidance and expertise in silicon design for testability.

11 References

[IEEE01] "IEEE Standard Test Access Port and Boundary-Scan Architecture", IEEE Std 1149.1-2001

[IEEE03] "IEEE Standard for Boundary-Scan Testing of Advanced Digital Networks", IEEE Std 1149.6-2003

[IEEEWG] IEEE 1149.1 Working Group website which contains meeting minutes and a private draft area: http://grouper.ieee.org/groups/1149/1/

[Park03] "The Boundary-Scan Handbook", 3rd Edition, Parker, K. P., Kluwer Academic Publishers (now Springer), Boston, MA, 2003

Table 1: The 'Interconnect' test process.

Step	Action	Why	Device 1	Device 2
1	TLR	Initialize TAP.	Garden variety Boundary-Scan device.	Garden variety Boundary-Scan device.
2	IRScan	Get ready for EXTEST.	Load PRELOAD	Load PRELOAD
3	DRScan	Set up first BReg pattern of test.	Shift test data into BReg, ignore data out.	Shift test data into BReg, ignore data out
4	IRScan	Switch to test mode.	Load EXTEST. This lobotomizes the device at Update-IR.	Load EXTEST. This lobotomizes the device at Update-IR.
5	DRScan	Set up next test pattern.	Shift test data into BReg, save result data out.	Shift test data into BReg, save result data out
6	Loop?	Multiple test patterns. Return to step 5.		
7	DRScan	Shift out last pattern result.	Shift safe data into BReg, save data out.	Shift safe data into BReg, save data out.
8	TLR	End of test, analyze result data.	I/O reconnected, but device still lobotomized.	I/O reconnected, but device still lobotomized.

Table 2: Interconnect test with an INITIALIZE preamble for 3 of 4 devices.

Step	Action	Why	Device 1	Device 2	Device 3	Device 4
1	TLR	Initialize TAP.	Garden variety Boundary-Scan device.	Has INIT_SETUP only.	Has INIT_RUN only.	Has both INIT_SETUP and INIT_RUN.
2	IRScan	Get ready for EXTEST.	Load PRELOAD.	Load PRELOAD.	Load PRELOAD.	Load PRELOAD.
3	DRScan	Set up first BReg pattern of test.	Shift test data into BReg, ignore data out.	Shift test data into BReg, ignore data out	Shift test data into BReg, ignore data out	Shift test data into BReg, ignore data out
Init1	IRScan	Get ready for setup data	Load BYPASS	Load INIT_SETUP	Load BYPASS	Load INIT_SETUP
Init2	DRScan	Load setup data		Shift Device2 setup data.		Shift Device4 setup data.
Init3	IRScan	Complete init process	Load EXTEST, controls outputs, lobotimization.	Load EXTEST, controls outputs, lobotimization.	Load INIT_RUN, outputs clamp, lobotimization.	Load INIT_RUN, outputs clamp, lobotimization.
Init4	DRScan	Read out status	Load in same data as step 3.	Load in same data as step 3.	Device3 status out	Device4 status out
Init5	Loop?	If not "Done" go to step Init4.			Check Device3 status.	Check Device4 status.
4	IRScan	Set up chain for test	Load EXTEST.	Load EXTEST.	Load EXTEST.	Load EXTEST.
5	DRScan	Set up next test pattern.	Shift test data into BReg, save result data out.	Shift test data into BReg, save result data out.	Shift test data into BReg, save result data out.	Shift test data into BReg, save result data out.
6	Loop?	Multiple test patterns. Return to step 5.				
7	DRScan	Shift out last pattern result.	Shift safe data into BReg, save data out.	Shift safe data into BReg, save data out.	Shift safe data into BReg, save data out.	Shift safe data into BReg, save data out.
8	TLR	End of test, analyze result data.	I/O reconnected, but still lobotomized.	I/O reconnected, but still lobotomized.	I/O reconnected, but still lobotomized.	I/O reconnected, but still lobotomized.

www.agilent.com
www.agilent.com/find/boundaryscan

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

www.axiestandard.org
AdvancedTCA® Extensions for Instrumentation and Test (AXIe) is an open standard that extends the AdvancedTCA® for general purpose and semiconductor test. Agilent is a founding member of the AXIe consortium.

www.lxistandard.org
LAN eXtensions for Instruments puts the power of Ethernet and the Web inside your test systems. Agilent is a founding member of the LXI consortium.

http://www.pxisa.org
PCI eXtensions for Instrumentation (PXI) modular instrumentation delivers a rugged, PC-based high-performance measurement and automation system.

Agilent Channel Partners

www.agilent.com/find/channelpartners
Get the best of both worlds: Agilent's measurement expertise and product breadth, combined with channel partner convenience.

Agilent Advantage Services

Agilent Advantage Services is committed to your success throughout your equipment's lifetime. We share measurement and service expertise to help you create the products that change our world. To keep you competitive, we continually invest in tools and processes that speed up calibration and repair, reduce your cost of ownership, and move us ahead of your development curve.

www.agilent.com/find/advantageservices

Agilent Electronic Measurement Group
KEMA Certified
ISO 9001:2008
Quality Management System

www.agilent.com/quality

For more information on Agilent Technologies' products, applications or services, please contact your local Agilent office. The complete list is available at:

www.agilent.com/find/contactus

Americas	
Canada	(877) 894 4414
Brazil	(11) 4197 3500
Mexico	01800 5064 800
United States	(800) 829 4444

Asia Pacific	
Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Other AP Countries	(65) 375 8100

Europe & Middle East	
Belgium	32 (0) 2 404 93 40
Denmark	45 70 13 15 15
Finland	358 (0) 10 855 2100
France	0825 010 700*
	*0.125 €/minute
Germany	49 (0) 7031 464 6333
Ireland	1890 924 204
Israel	972-3-9288-504/544
Italy	39 02 92 60 8484
Netherlands	31 (0) 20 547 2111
Spain	34 (91) 631 3300
Sweden	0200-88 22 55
United Kingdom	44 (0) 118 9276201

For other unlisted Countries:
www.agilent.com/find/contactus

Revised: October 14, 2010

Product specifications and descriptions in this document subject to change without notice.

Agilent Technologies