Testing Memories without Bios on the Latest Intel® Platforms

by Larry Osborn

ASSET

Larry Osborn

Larry Osborn, Senior Product Manager, at ASSET InterTech, has over 30 years of experience in product management, hardware/software product design and development, product delivery to the marketplace and user support. Over the years, Larry has a proven track record for identifying user needs and opportunities in the marketplace, providing innovative solutions and exceeding the expectations of users. At ASSET, Larry is responsible for the profit and loss for a product group. Prior to ASSET, he has held positions with Lockheed Martin, OCD Systems, Wind River, Hewlett-Packard, Ford Aerospace and Intel® Corporation. He holds a Bachelor's Degree in Computer Science from the University of Kansas and various technical and marketing training certifications.

Table of Contents

Current Testing Methods

Functional test is nothing new or glamorous and is often thought of as a necessary evil. The reasons for this concept of a "necessary evil" could be that every extra hour spent on getting the product to market has a cost associated with it. The focus today is on managing this investment to tighter constraints to maximize the return. The test development teams are getting smaller, and the test effort is being moved to earlier in the development cycle; some refer to this as "shift left". For Intel designs, some of this test effort has made its way into the BIOS. After all, flash memories are vast, and with a simple switch in the code, the product can test itself (at least that is the premise). The problem is that often the goal of booting the board is in conflict with the purpose of testing the board. Three areas are trending in manufacturing test for Intel Platforms, and this conflict is the source of problems associated with two of the three solutions. These "solutions" are the BIOS, Debug BIOS usage, and post-boot memory test utilities. This paper looks at the challenges associated with the use of each of these test "methodologies", examines their pitfalls and tries to answer the question, "Is there a better alternative?".

The Challenges

BIOS as a Test Method

The BIOS and the use of port 80h POST codes have long been used as a test mechanism. The software has progressed from the BIOS to UEFI to handle the complexity of the hardware and code requirements for booting the platform. Port 80h is still used to report errors. Consumers have watched, indirectly, the code growth of the BIOS with longer and longer boot times. Not surprisingly, the BIOS designers have continually been pummeled with complaints that boot time is too long. So now, every effort is made to boot quickly and at all costs. After all, from a BIOS developer's perspective, the hardware is probably good. This viewpoint can lead the BIOS developer to ignore problems with the training of memory and channel controller initialization and other hardware issues in the platform. In some cases, the obfuscation of the hardware issues is a side-effect to achieve the goal of minimal boot time.

Debug BIOS as a Test Method

An alternative is to use the software DEBUG switch within the BIOS to create a test method. This DEBUG switch too can be problematic for use in production testing. First, the software

makes assumptions about the health of the hardware. The test portion of the DEBUG BIOS software must produce diagnostic messages in case of hardware failure. These messages are often targeted at the console. If the console did not get initialized, then the old reliable POST code LEDs are always available, but they are a poor substitute for problem diagnosis. The diagnostic granularity for this DEBUG switch also makes assumptions about the availability of time, and in a production setting, there simply isn't enough time to execute the DEBUG BIOS and meet a production beat rate schedule. So getting a good diagnostic resolution provided by the DEBUG BIOS, in a production setting, simply is not financially feasible.

Post-Boot Test Utilities

Contract Manufacturers (CM) will often use the booting of the UUT as an indication that the platform is ready for further testing. Then a utility like memtest86 is launched to test the platform, beyond what the BIOS has "already tested". So, beyond what has previously mentioned about the BIOS challenges, what could be the problem with these utilities? One challenge is the additional time for the board to boot before more testing can begin. The boot time increases the beat rate on the production floor which is nearly always considered a negative. There are other challenges like if a blank screen or blue screen occurs. When that occurs, the board is then shifted to the bone pile where the technicians can use the Port 80h POST Codes to begin a shotgun approach to testing, remove devices, or scrap the PCB.

It is clear that there are pitfalls associated with using these three methods (BIOS, DEBUG BIOS, post-boot) as part of the test methodology. Finally, could some of these dependencies on using the BIOS be the culprit in the increase of RMA's and product quality issues trending in today's marketplace?

Can We Find a Better Way?

Manufacturing Test without BIOS or DEBUG BIOS

From a production test standpoint, the best solution would be free from the need of the BIOS and free from the requirement that particular platform hardware features, like the console, must be operational. But is being free from the BIOS constraint even possible? A very high-level view of the BIOS execution can provide some insight as to what is needed. Early stages of the BIOS provide for device initialization done through writes to registers. The BIOS does contain the

Memory Reference Code (MRC), which provides for the initialization of memory devices and the training of the memory devices to operate at the most efficient performance level. After some initial device initialization, the MRC is executed to get memory functional, so the rest of the boot process has RAM to run. From then on, it is a matter of loading DXE drivers and to prepare to load the OS.

If there were a means of knowing what registers to write to and in what sequence, then the need for the BIOS could be restricted to the MRC execution.

Breaking down the challenges

Tackling the first problem; what registers to write and the correct sequence.

Most CM's have or are provided with at least one known golden board. By reading the registers of the processor and/or chipset from the known working board, it would be possible to store the last known values. Using these recorded values as a source for a "player" that initializes a target platform under test would simulate a correct BIOS setup. A prerequisite for this approach to work would be that the board configuration (device population) is the same. Many platform designs are derived from an Intel Customer Reference Board (CRB), so it should be possible to use that CRB as the gold standard and use this platform setup as the source for the production test player, albeit with a few modifications to match the derivative. So it is possible.

The second challenge*; what about writing register sequences?*

The source for the player would not have knowledge of these write sequence dependencies. The sequence dependencies provide a bit of a complication. Some devices require a particular sequence of writes to a single or to multiple registers to complete the initialization sequence. This unique series requires the need to have some intelligence applied based upon the device specification. By using the device id, it would be possible to know when a playback is writing to a device that requires a sequence written. Then what? The best approach would be a model-based, where the device model knows the correct sequence of registers to initialize the device. Then just apply the pattern to the device. But there needs to be a consideration of what knowledge the model has versus the particular implementation on the UUT. So a merging of the data from the known working board with the model is required. Not an easy problem, but also doable.

Memory Reference Code challenge.

The most involved question is how to manage the memory setup or the MRC. There are two potential choices. First is to run the MRC code, supplied in the BIOS in object form, which begs the question of how to get only the MRC object loaded. Or option two is to get the MRC sources, BIOS developer license required, then compile and load them. But, there is one additional complication with the MRC execution: the MRC must complete or the memory devices will not be accessible for the test execution. So the manifestation of this last problem requires that the MRC be instrumented in some fashion to stop on a setup failure and report the failure. To discuss all the instrumentation required within the MRC is beyond the scope of this paper. The MRC topic discussed here is intended to make the reader aware of the magnitude of the work involved, and also to confirm that it is doable.

Increasing the diagnostic resolution

In the model described here, the discussion has been essentially about device or platform setup. What is missing is to address the question, "How to test?" Testing from a functional viewpoint requires that endpoints provide feedback to the tests. These endpoints could be registers within the device or the response from the device (i.e. SATA) to acknowledge setup is complete. For memory testing, once the MRC is complete, a series of memory test algorithms can be employed. Platform knowledge is required within the models as the devices interact based upon specified behaviors.

A platform model could provide the perfect opportunity to include verification of setup values written to the UUT and embedded error message with full diagnostic data. As mentioned earlier, some devices require patterns or sequences be written. The sequence necessitates that some scripts or logic be applied, and that is where diagnostic messages can be embedded. Again, the platform model is a logical store point for these actions. The diagnostic messages can be tailored to provide the maximum diagnostic resolution based upon the given test. With the platform model providing the tests, this model can be merged with the device model, and together they constitute a test profile unique to a UUT.

Recapping the Challenge

Recapping the inherent issues for using BIOS as a test vehicle are:

1. Engineering goal conflict. Boot at all costs vs. testing. Hiding memory from the test.
2. Assumption that hardware is functional
3. Blue Screen or No Screen
4. MRC must complete
5. A lot of development work require for model development, MRC instrumentation, data extraction and so on.

It is clear that there is a need for some solution besides using UUT resources to provide testing of the UUT. And there is the possibility that all challenges can be met. What might not be so obvious is that an external host is required to provide the platform model, and run the MRC and test engine. The external host also provides for the display of the diagnostic messages and data logging of the tests conducted on the UUT.

Is There a Solution?

There is a solution to these problems, and it is provided by ASSET InterTech with the ScanWorks® platform and its Processor-Controlled Test solution. The product provides for automated test development of both Core® and Xeon® platforms including an instrumented MRC and UUT/device model library. By using JTAG and an external host, it is possible to adequately test an Intel design without using the BIOS. The challenges are different between Core and Xeon platform and more detail on this topic will be provided in a future white paper.

Learn More

Learn more about Processor-Controlled Test products on our website.