15 Testing, Debugging, and Verification

15.1 Introduction

While in real estate the refrain is “Location! Location! Location!” the comparable advice in IC design should be “Testing! Testing! Testing!” For many chips, testing accounts for more effort than does design.

Tests fall into three main categories. The first set of tests verifies that the chip performs its intended function. These tests, called *functionality tests* or *logic verification*, are run before tapeout to verify the functionality of the circuit. The second set of tests are run on the first batch of chips that return from fabrication. These tests confirm that the chip operates as it was intended and help debug any discrepancies. They can be much more extensive than the logic verification tests because the chip can be tested at full speed in a system. For example, a new microprocessor can be placed in a prototype motherboard to try to boot the operating system. This *silicon debug* requires creative detective work to locate the cause of failures because the designer has much less visibility into the fabricated chip compared to during design verification. The third set of tests verify that every transistor, gate, and storage element in the chip functions correctly. These tests are conducted on each manufactured chip before shipping to the customer to verify that the silicon is completely intact. These are called *manufacturing tests*. In some cases, the same tests can be used for all three steps, but often it is better to use one set of tests to chase down logic bugs and another, separate set optimized to catch manufacturing defects.

In Section 14.5.2, we noted that the yield of a particular IC was the number of good die divided by the total number of die per wafer. Because of the complexity of the manufacturing process, not all die on a wafer function correctly. Dust particles and small imperfections in starting material or photomasking can result in bridged connections or missing features. These imperfections result in what is termed a *fault*. Later in the chapter, we will examine a number of fault mechanisms. The goal of a manufacturing test procedure is to determine which die are good and should be shipped to customers.

Testing a die (chip) can occur at the following levels:

- Wafer level
- Packaged chip level
- Board level
- System level
- Field level

By detecting a malfunctioning chip early, the manufacturing cost can be kept low. For instance, the approximate cost to a company of detecting a fault at the various levels [Williams86] is at least

Wafer	\$0.01–\$0.10
Packaged chip	\$0.10–\$1
Board	\$1–\$10
System	\$10–\$100
Field	\$100–\$1000

Obviously, if faults can be detected at the wafer level, the cost of manufacturing is lower. In an extreme example, Intel failed to correct a logic bug in the Pentium floating-point divider until more than 4 million units had shipped in 1994. IBM halted sales of Pentium-based computers and Intel was forced to recall the flawed chips. The mistake and lack of prompt response cost the company an estimated \$450 million.

It is interesting to note that most failures of first-time silicon result from problems with the functionality of the design; i.e., the chip does exactly what the simulator said it would do, but for some reason (almost always human error) this functionality is not what the rest of the system expects.

The remainder of this section will provide an overview of the processes involved in logic verification, chip debug, and manufacturing test. Section 15.2 discusses the mechanics of testing and test programs. Sections 15.3 through 15.5 address the principles behind each phase of testing. If testing is not considered in advance, the manufacturing test can be extremely time consuming and hence expensive. Some chips have even proved impossible to debug because designers have so little visibility into the internal operation. Sections 15.6 and 15.7 focus on how to design chips to facilitate debug and manufacturing test at the chip and board level. [Wang08b] offers an entire book dedicated to test.

15.1.1 Logic Verification

Verification tests are usually the first ones a designer might construct as part of the design process. Does this adder add? Does this counter count? Does this state-machine yield the right outputs each cycle? Does this modem decode data correctly?

In Section 14.4.1.3, we noted that verification tests were required to prove that a synthesized gate description was functionally equivalent to the source RTL. Figure 15.1 shows that we may want to prove that the RTL is equivalent to the design specification at a higher behavioral or specification level of abstraction. The behavioral specification might be a verbal description, a plain language textual specification, a description in some high-level computer language such as C, a program in a system-modeling language such as SystemC, or a hardware description language such as VHDL or Verilog, or simply a table of inputs and required outputs. Often, designers produce a *golden model* in one of the previously mentioned formats and it becomes the reference against which all other representations are checked. Functional equivalence involves running a simulator on the two descriptions of the chip (e.g., one at the gate level and one at a functional level) and ensuring that the outputs are equivalent at some convenient check points in time for all inputs applied. This is most conveniently done in an HDL by employing a *test bench*; i.e., a wrapper that surrounds a module and provides for stimulus and automated checking. The most detailed check might be on a cycle-by-cycle basis. Increasingly, verification involves real-time or near real-time emulation in an FPGA-based system to confirm system-level performance *in situ*; i.e., in the actual system that will use the end chip. This is recommended because of the increasing level of complexity of chips and the systems they implement. As an example, in the area of wireless local area network chips, without a real-time emulation system, it is virtually impossible to simulate the unseen effects of an unreliable channel with out-of-band interferers.

You can check functional equivalence through simulation at various levels of the design hierarchy. If the description is at the RTL level, the behavior at a system level may be able to be fully verified. For instance, in the case of a microprocessor, you can boot the operating system and run key programs for the behavioral description. However, this might be impractical (due to long simulation times) for a gate-level model and even harder for a transistor-level model. The way out of this impasse is to use the hierarchy inherent within a system to verify chips and modules within chips. That, combined with well-defined modular interfaces, goes a long way in increasing the likelihood that a system composed of many VLSI chips will be first-time functional.

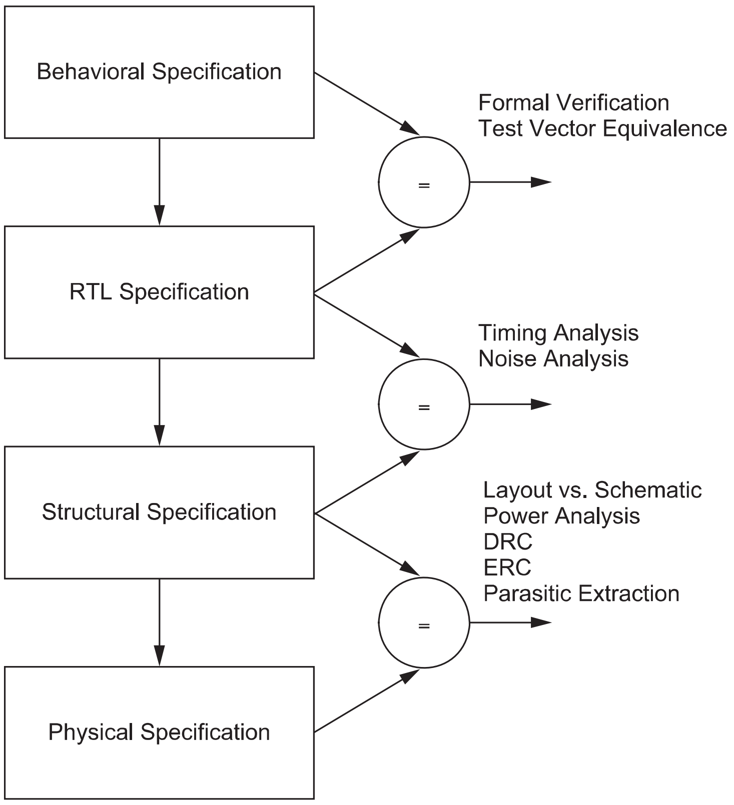

FIGURE 15.1 Functional equivalence at various levels of abstraction

The best advice with respect to writing functional tests is to simulate as closely as possible the way in which the chip or system will be used in the real world. Often, this is impractical due to slow simulation times and extremely long verification sequences. One approach is to move up the simulation hierarchy as modules become verified at lower levels. For instance, you could replace the gate-level adder and register modules in a video filter with functional models and then in turn replace the filter itself with a functional model. At each level, you can write small tests to verify the equivalence between the new higher-level functional model and the lower-level gate or functional level. At the top level, you can surround the filter functional model with a software environment that models the real-world use of the filter. For instance, you can feed a carefully selected subsample of a video frame to the filter and compare the output of the functional model with what the designer expected theoretically. You can also observe the video output on a video frame buffer to check that it looks correct (by no means an exhaustive test, but a confidence builder). Finally, if enough time is available, you can apply all or part of the functional test to the gate level and even the transistor level if transistor primitives have been used.

Verification at the top chip level using an FPGA emulator offers several advantages over simulation and, for that matter, the final chip implementation. Most noticeably, the emulation speed can be near if not real time. This means that the actual analog signals (if used) can be interfaced with the chip. Additionally, to assess system performance, you can introduce fine levels of observation and monitoring that might not be included in the final chip. For instance, you could include a bit-error rate circuit in a communication modem to aid performance optimization.

In most projects, the amount of verification effort greatly exceeds the design effort. Remember the following statement, culled from many years of IC design experience, whenever you are tempted to minimize verification effort to meet tight schedules: “If you don’t test it, it won’t work! (guaranteed).”

15.1.2 Debugging

Many times, when a chip returns from fabrication, the first set of tests are run in a lab environment, so you need to prepare for this event. You can begin by constructing a circuit board that provides the following attributes:

- Power for the IC with ability to vary V_{DD} and measure power dissipation
- Real-world signal connections (i.e., analog and digital inputs and outputs as required)
- Clock inputs as required (it is helpful to have a stable variable-frequency clock generator)
- A digital interface to a PC (either serial or parallel ports for slow data or PCI bus for fast data interchanges)

You can write software routines to interface with the chip through the serial or parallel port or the bus interface. The chip should have a serial UART port or some other interface that can be used independently of the normal operation of the chip. The lowest level of the software should provide for peeking (reading) and poking (writing) registers in the chip. An alternate or complementary approach is to provide interfaces for a logic analyzer. These are easily added to a PCB design in the form of multipinned headers. Figure 15.2 shows a typical test board, illustrating the *zero insertion force* (ZIF) socket for the chip (in the center of the board), an area for analog circuitry interface (on the left), a set of headers for logic analyzer connection (at the top and bottom) and a set of programmable power supplies (on the right). In addition, an interface is provided for control by a serial port of a PC (at the bottom left).

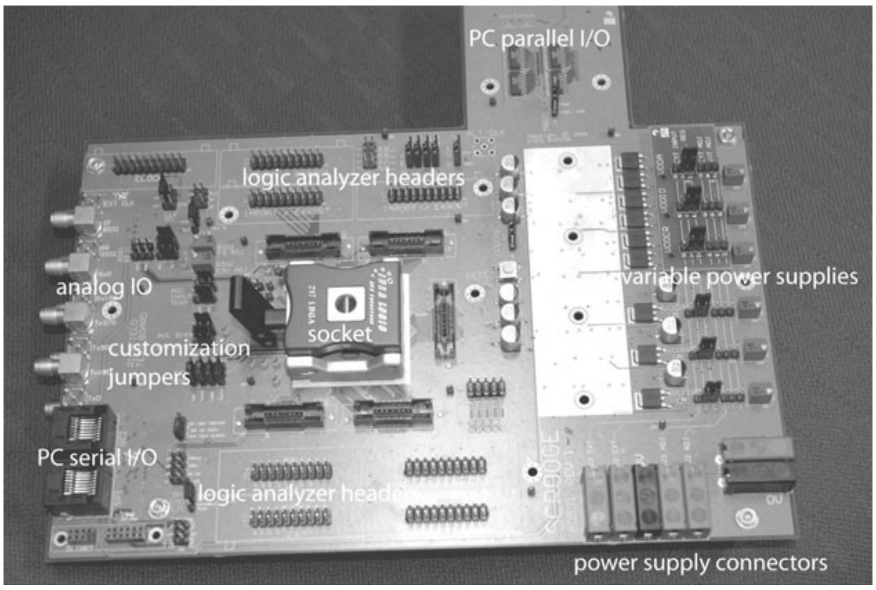

FIGURE 15.2 Typical test board

You should start with a “smoke test.” This involves ramping the supply voltages from zero to V_{DD} while monitoring the current without any clocks running. For a fully static circuit, the current should remain at zero. Analog circuits will draw their quiescent current.

Following this, you can enable the clock(s); some dynamic current should be evident. Beware that many CMOS chips appear to operate when the clock is connected but the power supply is turned off because the clock may partially power the chip through the input protection diodes on the input pads. If possible, you should initially run the clock at reduced speed so that setup time failures are not the initial culprit in any debug operation.

In the case of a digital circuit, you should examine various registers for health using PC-based peek and poke software. This checks the integrity of the signal path from the PC to the chip. Often, designers place an ID in the register at address zero. Peeking at this register proves the read path from the chip. If the chip registers are reset to a known state, the registers can be read sequentially and compared with the design values. In the case of the logic analyzer, you can download the equivalent test pattern to exercise the chip. Frequently, these patterns can be automatically generated from the verification test bench. Up to this point, no functionality of the chip has been exercised apart from register reads and writes.

Where the chip has built-in self-test (see Sections 15.6 and 15.7), you can run the commercial software that provides for this functionality over a boundary scan interface. This type of system automatically runs a set of tests on the chip that completely verify the correct operation of all gates and registers as defined by the original RTL description. If this kind of a test interface was not used, you should pursue a manual effort in which the functionality of the chip is checked from the bottom-up. Of course, if you are a gambler, you can do a top-level test like running a piece of code or trying to boot the operating system right away. Experience shows that this often does not work, usually because of problems with the test fixture, and so you must revert to the bottom-up method to prove that one piece of the design works at a time.

If you detect anomalous behavior, you must go about debugging. The basic method is to postulate a method of failure, then test the hypothesis. Debug is an art in itself, but some pointers for sane debugging are as follows:

- Keep an annotated and dated logbook for all tests done.
- When postulating a cause for the bug and a test, do one change at a time and observe the result: Changing many things and then seeing if they work will not logically lead you to the bug and is commonly called the “shotgun approach.”
- Check everything two or three times; never assume anything unless it is measured and logged in a notebook. Have someone independently check critical measurements.
- Check signals and supply voltages at the pins of the IC; frequently, new test boards have errors.
- Double-check the specified chip I/O and perform a continuity check from the IC pins to expected places (i.e., test pins, supplies) on the board.
- Never disregard a possible reason for a bug, however crazy, unless you can prove it isn’t the cause.
- Use freeze spray or a heat gun to cool down or heat up a circuit to check for temperature problems.
- Check the state of any internal registers against that noted in the documentation.
- Evaluate the timing of any inputs and outputs with respect to the clock; often setup or hold times can be violated in a new test setup.

- When a bug is discovered and corrected, hunt for other portions of the design that might have a similar bug that hasn't been detected yet. Where there is one rat, there are many rats!
- Never assume anything—question everything—a slight touch of paranoia helps!!

[Agans06] cites nine "debugging rules" that bear repeating:

- *Understand the system.* If you are the designer, this should be self-evident. However, if you have been assigned to the task of debugging, follow this point keenly.
- *Make it fail.* Find a way to elicit the bug. A repeatable method is preferable.
- *Quit thinking and look.* Propose a test and investigate. You can start to eliminate possible sources of problems.
- *Divide and conquer.* Use hierarchy to eliminate known good parts of the system.
- *Change one thing at a time.* A very important rule.
- *Keep an audit trail.* No matter how good your memory is, a written account serves as a memory jog and something for someone else to look at to propose approaches.
- *Check the plug.* Check the complete test structure. More problems are found in new test harnesses than in the actual chip due to the level of verification used in each.
- *Get a fresh view.* Get a coworker involved. Take a break. Take a nap.
- *If you didn't fix it, it ain't fixed.* Problems do not mysteriously fix themselves. If you find a problem, verify it with simulation to prove your hypothesis of the failure mode.

After the chip is demonstrated to be operational, you can measure more subtle aspects of the design such as performance (power, speed, analog characteristics). This involves normal lab techniques of configure, measure, and record. Where possible, store all results as computer readable results (i.e., stored images from digital oscilloscope and screen dumps from logic analyzer) for communication with colleagues.

For the most part, if a digital chip simulates at the gate level and passes timing analysis checks during design, it will do exactly the same in silicon. Possible deviations from the simulated circuit occur in the following cases:

- Circuit is slower than predicted—fix—slow clock or raise V_{DD}
- Circuit has a race condition—fix—heat with heat gun if a logic gate caused race
- Circuit has dynamic logic problems—fix—don't do it again
- Gnarly crosstalk problems—fix—get better tools
- Wrong functionality—fix—do a better job of verification

With analog circuitry, a wide range of issues can affect performance over and above what was simulated. These include power and ground noise, substrate noise, and temperature and process effects. However, you can employ the same basic debug approaches.

15.1.3 Manufacturing Tests

Whereas verification or functionality tests seek to confirm the function of a chip as a whole, manufacturing tests are used to verify that every gate operates as expected. The need to do this arises from a number of manufacturing defects that might occur during either chip fabrication or accelerated life testing (where the chip is stressed by over-voltage and over-temperature operation). Typical defects include the following:

- Layer-to-layer shorts (e.g., metal-to-metal)
- Discontinuous wires (e.g., metal thins when crossing vertical topology jumps)
- Missing or damaged vias
- Shorts through the thin gate oxide to the substrate or well

These in turn lead to particular circuit maladies, including the following:

- Nodes shorted to power or ground
- Nodes shorted to each other
- Inputs floating/outputs disconnected

Tests are required to verify that each gate and register is operational and has not been compromised by a manufacturing defect. Tests can be carried out at the wafer level to cull out bad dies, or can be left until the parts are packaged. This decision would normally be determined by the yield and package cost. If the yield is high and the package cost low (i.e., a plastic package), then the part can be tested only once after packaging. However, if the wafer yield was lower and the package cost high (i.e., an expensive ceramic package), it is more economical to first screen bad dice at the wafer level. The length of the tests at the wafer level can be shortened to reduce test time based on experience with the test sequence.

Apart from the verification of internal gates, I/O integrity is also tested, with the following tests being completed:

- I/O levels (i.e., checking noise margin for TTL, ECL, or CMOS I/O pads)
- Speed test

With the use of on-chip test structures described in Section 15.6, full-speed wafer testing can be completed with a minimum of connected pins. This can be important in reducing the cost of the wafer test fixture.

In general, manufacturing test generation assumes the function of the circuit/chip is correct. It requires ways of exercising all gate inputs and monitoring all gate outputs.

Example 15.1

Consider testing the MIPS microprocessor from Chapter 1. Explain the difference between the tests you would use for logic verification or silicon debug and the tests you would use for manufacturing.

SOLUTION: Logic verification should test that each operation can be performed. For example, a test program might exercise all of the instructions to demonstrate that each one behaves as intended. Logic verification will not necessarily prove that the instruction works for all possible addresses and data values. In contrast, manufacturing tests must prove that every gate operates correctly. They ideally stimulate each gate to produce both a 0 and a 1 to ensure the gate is not damaged. The manufacturing tests may be the only tests applied to a microprocessor prior to it being placed in a system and used. Clearly, it is a challenge to devise a set of tests that is both complete enough that customers receive very few defective chips and short enough to keep testing economical.