

FUTURE VISION BIE

One Stop for All Study Materials
& Lab Programs



Future Vision

By K B Hemanth Raj

Scan the QR Code to Visit the Web Page



Or

Visit : <https://hemanthrajhemu.github.io>

Gain Access to All Study Materials according to VTU,
CSE – Computer Science Engineering,
ISE – Information Science Engineering,
ECE - Electronics and Communication Engineering
& MORE...

Join Telegram to get Instant Updates: https://bit.ly/VTU_TELEGRAM

Contact: MAIL: futurevisionbie@gmail.com

INSTAGRAM: www.instagram.com/hemanthraj_hemu/

INSTAGRAM: www.instagram.com/futurevisionbie/

WHATSAPP SHARE: <https://bit.ly/FVBIESHARE>

CMOS VLSI Design

A Circuits and Systems Perspective

Fourth Edition

Neil H. E. Weste

*Macquarie University and
The University of Adelaide*


David Money Harris

Harvey Mudd College

Addison-Wesley

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

<https://hemanthrajhemu.github.io>

14.6	Data Sheets and Documentation	655
14.6.1	The Summary	655
14.6.2	Pinout	655
14.6.3	Description of Operation	655
14.6.4	DC Specifications	655
14.6.5	AC Specifications	656
14.6.6	Package Diagram	656
14.6.7	Principles of Operation Manual	656
14.6.8	User Manual	656
 14.7	CMOS Physical Design Styles	656
14.8	Pitfalls and Fallacies	657
Exercises		657

Chapter 15 Testing, Debugging, and Verification

15.1	Introduction	659
15.1.1	Logic Verification	660
15.1.2	Debugging	662
15.1.3	Manufacturing Tests	664
15.2	Testers, Test Fixtures, and Test Programs	666
15.2.1	Testers and Test Fixtures	666
15.2.2	Test Programs	668
15.2.3	Handlers	669
15.3	Logic Verification Principles	670
15.3.1	Test Vectors	670
15.3.2	Testbenches and Harnesses	671
15.3.3	Regression Testing	671
15.3.4	Version Control	672
15.3.5	Bug Tracking	673
15.4	Silicon Debug Principles	673
15.5	Manufacturing Test Principles	676
15.5.1	Fault Models	677
15.5.2	Observability	679
15.5.3	Controllability	679
15.5.4	Repeatability	679
15.5.5	Survivability	679
15.5.6	Fault Coverage	680
15.5.7	Automatic Test Pattern Generation (ATPG)	680
15.5.8	Delay Fault Testing	680
15.6	Design for Testability	681
15.6.1	<i>Ad Hoc</i> Testing	681
15.6.2	Scan Design	682
15.6.3	Built-In Self-Test (BIST)	684
15.6.4	IDDQ Testing	687
15.6.5	Design for Manufacturability	687



15.7	Boundary Scan	688
15.8	Testing in a University Environment	689
15.9	Pitfalls and Fallacies	690
	Summary	697
	Exercises	697

Appendix A Hardware Description Languages

A.1	Introduction	699
A.1.1	Modules	700
A.1.2	Simulation and Synthesis	701
A.2	Combinational Logic	702
A.2.1	Bitwise Operators	702
A.2.2	Comments and White Space	703
A.2.3	Reduction Operators	703
A.2.4	Conditional Assignment	704
A.2.5	Internal Variables	706
A.2.6	Precedence and Other Operators	708
A.2.7	Numbers	708
A.2.8	Zs and Xs	709
A.2.9	Bit Swizzling	711
A.2.10	Delays	712
A.3	Structural Modeling	713
A.4	Sequential Logic	717
A.4.1	Registers	717
A.4.2	Resettable Registers	718
A.4.3	Enabled Registers	719
A.4.4	Multiple Registers	720
A.4.5	Latches	721
A.4.6	Counters	722
A.4.7	Shift Registers	724
A.5	Combinational Logic with Always / Process Statements	724
A.5.1	Case Statements	726
A.5.2	If Statements	729
A.5.3	SystemVerilog Casez	731
A.5.4	Blocking and Nonblocking Assignments	731
A.6	Finite State Machines	735
A.6.1	FSM Example	735
A.6.2	State Enumeration	736
A.6.3	FSM with Inputs	738
A.7	Type Idiosyncracies	740

Testing, Debugging, and Verification

15

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.

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.

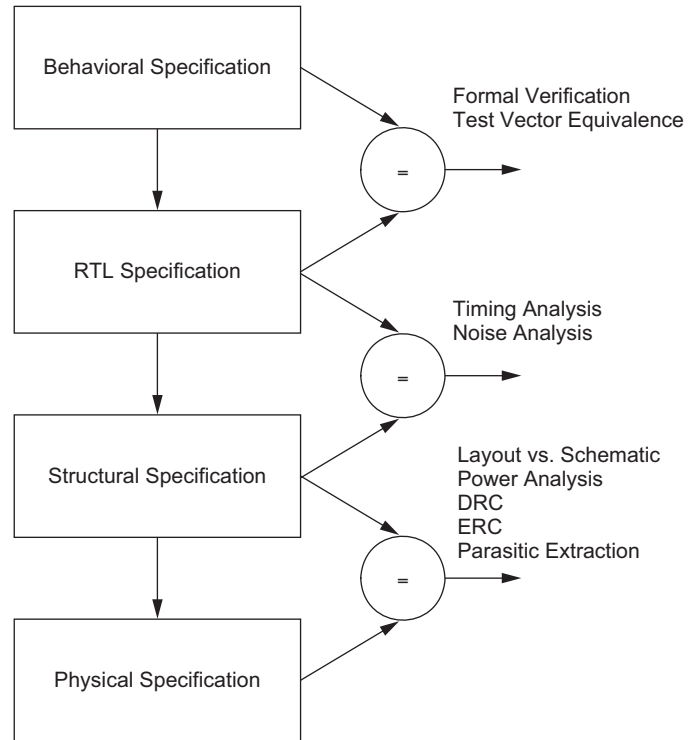


FIGURE 15.1 Functional equivalence at various levels of abstraction

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).

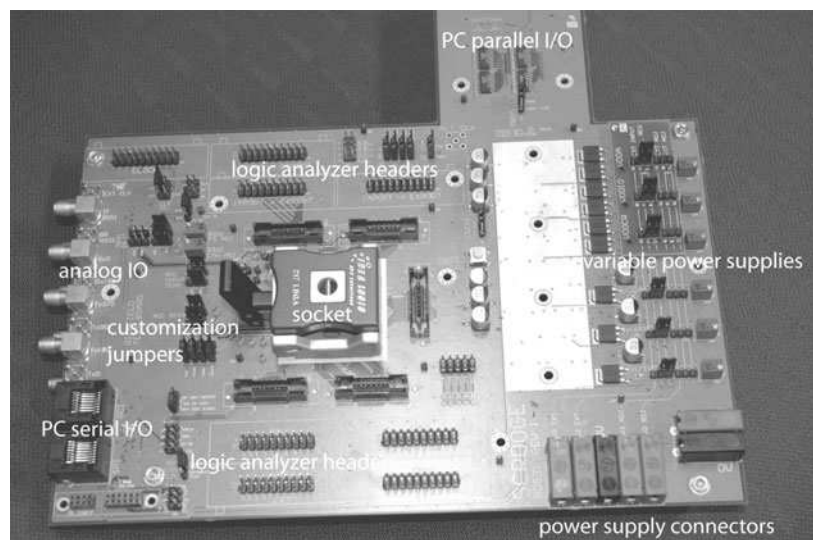


FIGURE 15.2 Typical test board

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.

15.2 Testers, Test Fixtures, and Test Programs

To test a chip after it is fabricated, you need a tester, a test fixture, and a test program.

15.2.1 Testers and Test Fixtures

A *tester* is a device that can apply a sequence of stimuli to a chip or system under test and monitor and/or record the results of those operations. Testers come in various shapes and sizes.

To test a chip, one or more of four general types of *test fixtures* may be required. These are as follows:

- A *probe card* to test at the wafer level or unpackaged die level with a chip tester
- A *load board* to test a packaged part with a chip tester
- A printed circuit board (PCB) for bench-level testing (with or without a tester)
- A PCB with the chip *in situ*, demonstrating the system application for which the chip is used

We will concentrate first on the cases where a general-purpose production tester is to be used. Production testers are usually expensive pieces of equipment with configurable I/O ports (drive current, output levels, input levels) and huge amounts of RAM behind each test pin. The tester drives input pins from this memory on a cycle-by-cycle basis and samples and stores the levels on output pins. Figure 15.3 shows a typical production tester. In the background, you can see the four-bay cabinet holding the drive electronics. To the right in the background is the controlling workstation. The test head is shown on the front center. This is where the chip is placed in the load board to be tested.



FIGURE 15.3 The Teradyne Catalyst: A typical production tester (Photo: John Haddy, Cisco Systems.)

The probe card or load board for the *device under test* (DUT) is connected to the tester, as shown in Figure 15.4. The test program is compiled and downloaded into the tester and the tests are applied to the bare die or packaged chip. The tester samples the chip outputs and compares the values with those provided by the test program. If there are any differences, the chip is marked as faulty (with an ink dot) and the failing tests may be displayed for reference and stored for later analysis. In the case of a probe card, the card is raised, moved to the next die on the wafer, lowered, and the test procedure repeated. In the case of a load board with automatic part handling, the tested part is removed from the board and sorted into a good or bad bin. A new part is fed to the load board and the test is repeated. In most cases, these procedures take a few seconds for each part tested.

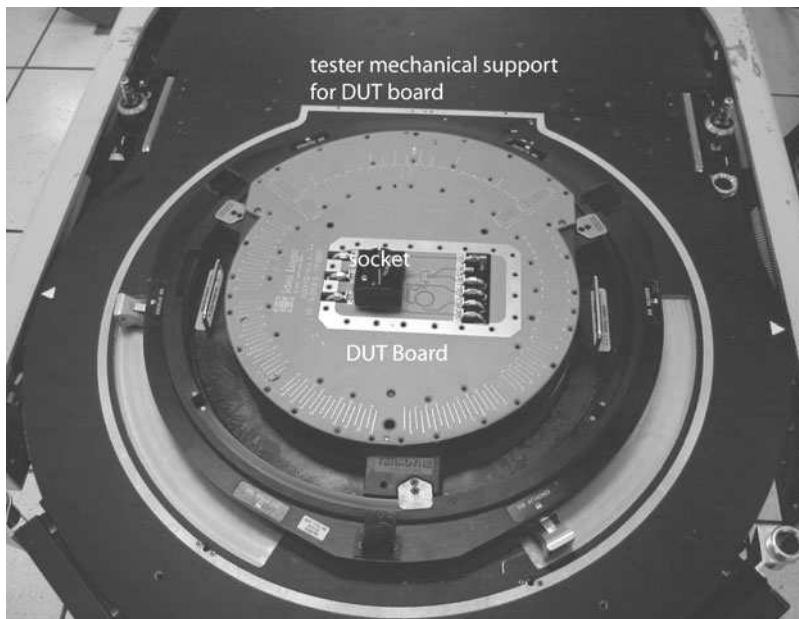


FIGURE 15.4 Tester load board in test head (Photo: John Haddy, Cisco Systems.)

The ability to vary the voltage and timing on a per-pin basis with a tester allows a process known as “shmooing” to be carried out. For instance, you could sweep V_{DD} from 3 V to 6 V on a 5 V part while varying the tester cycle time. This yields a graph called a *shmoo plot* that shows the speed sensitivity of the part with respect to voltage. Another shmoo that is frequently performed is to skew the timing on inputs with respect to the chip clock to look for setup and hold variations. Examples of shmoo plots and their interpretations are given in Section 15.4.

Testers can be very expensive, especially for high-frequency and/or analog/RF chips. Tester usage is charged by time, so the shorter a test runs, the cheaper a part is to test. Applying tests to check every node on the chip may be prohibitively costly, so some designs face a trade-off between test cost and the fraction of defective chips that slip through testing.

Example 15.2

Suppose a \$5 million tester has an expected useful life of two years before it becomes inadequate to test faster next-generation parts. How much does the tester cost per second?

SOLUTION: Dividing the tester cost by the number of seconds in two years gives \$0.08/second.

Testers are available that can be used to test an IC in a laboratory environment. They mirror large production testers, but generally have less functionality (e.g., slower, less memory per pin, less expandability) and are markedly less expensive. A probe card that allows wafer probing or a socketed load board is required for each design. A good logic analyzer with a pattern generator and a socketed test board can also be used to test a chip. Some groups effectively design their own logic analyzers by surrounding a chip with FPGAs and using the logic and RAM within the FPGA to apply and observe test patterns.

15.2.2 Test Programs

The tester requires a *test program* (in verification and test, this is an overloaded term). This program is normally written in a high-level language (for instance, the IMAGE language used by Teradyne is based on C) that supports a library of primitives for a particular tester. The test program specifies a set of input patterns and a set of output *assertions*. If an output does not match the asserted value at the corresponding time, the tester will report an error. Before the patterns and assertions are applied, the test program has to set up the various attributes of a tester such as the following:

- Set the supply voltages
- Assign mapping between stimulus file signal names and physical tester pins
- Set the pins on the tester to be inputs or outputs and their V_{OH}/V_{IH} levels
- Set the clock on the tester
- Set the input pattern and output assertion timing

And then on a per chip basis:

- Apply supply voltages
- Apply digital stimulus and record responses
- Check responses against assertions
- Report and log errors

A stimulus or pattern file can be derived from running a simulation on the design. Special *vector change descriptions* (VCDs) are used to compact simulation results. An example of a simple stimulus/pattern file for the case of a full adder follows:

	III	OO
		SC
		UA
		MR
		R
	ABC	Y
0	000	00
1	001	10
2	010	10
3	011	01
4	100	10
5	101	01
6	110	01
7	111	11

The first line designates the signal directions and shows three inputs (I) and two outputs (O). Reading downward, the next five lines designate the signal names (A, B, C, SUM, CARRY). Thereafter, each line designates a new *test vector*. The first column is the test vector number. The next three columns are the binary value of the inputs and the following two columns are the expected output values. Each line represents a certain length clock cycle that is asserted by the tester. Signals change after a specified period in relation to an internal clock running at the required test period. Clock generation can be carried out in two different ways. First, the clock can be treated like any other signal, in which case, it takes two tester cycles to complete a single clock cycle: one for the clock low and one for the clock high. Alternatively, a timing generator can be used, which allows the clock rising edge (for instance) to be placed anywhere in the tester cycle. So for instance, if the inputs are changed at the start of the tester cycle, the clock might be programmed to rise at the middle of the cycle.

Each pin on the tester is connected to a function memory, which is used to either drive an input or check an output at a DUT pin. Multiple bits may be required per pin to control tristate input pins or mask outputs when they should be ignored.

The clock speed, T_c , is specified, as are supply voltage levels. The time at which pins are driven and sampled is also specified on a pin-by-pin basis (T_j). The format of the test data is usually chosen from Non Return to Zero (NRZ), Return To Zero (RTZ), or other formats such as Surround By Zero (SBZ).

15.2.3 Handlers

An IC handler is responsible for feeding ICs to a test fixture attached to a tester. Chutes or trays containing packaged chips can be used to gravity-feed the devices to the handler, which uses a variety of mechanical means to pick the chips up and place them in the test socket on the load board. The tester stimulus is then applied and chips are binned depending on whether or not they passed the test. It is possible to heat and cool a chuck to test the chip at temperature. However, package-level testing is not normally carried out at temperature because of the time it takes to temperature-cycle the chuck.

An example of a handler is shown in Figure 15.5. This is the NS-6040 from Seiko-Epson. The body of the machine holds the mechanical positioning equipment, while the upper central section



FIGURE 15.5 Photograph of an Epson NS-6040 IC handler (Photo: John Haddy, Cisco Systems.)

supports the test fixture. The light on top indicates a functioning or stopped machine and is designed to be visible across a production floor where many machines might be operating. A screen at the top right provides status information to the operator. The unit has wheels for easy movement, but also has firm footings, which are lowered when the machine is in use.

Handlers add a constant time to the test process, typically around 1 second. Thus, load boards and handlers are often constructed to deal with two or four chips at once to reduce the cost of testing. Because a load board must be designed to fit to a given handler, select the handler before starting design of the load board.

15.3 Logic Verification Principles

Figure 15.6(a) shows a combinational circuit with N inputs. To test this circuit exhaustively, a sequence of 2^N inputs (or test vectors) must be applied and observed to fully exercise the circuit. This combinational circuit is converted to a sequential circuit with addition of M registers, as shown in Figure 15.6(b). The state of the circuit is determined by the inputs and the previous state. A minimum of 2^{N+M} test vectors must be applied to exhaustively test the circuit. As observed by [Williams83] more than two decades ago,

With LSI, this may be a network with $N = 25$ and $M = 50$, or 2^{75} patterns, which is approximately 3.8×10^{22} . Assuming one had the patterns and applied them at an application rate of $1 \mu\text{s}$ per pattern, the test time would be over a billion years (10^9).

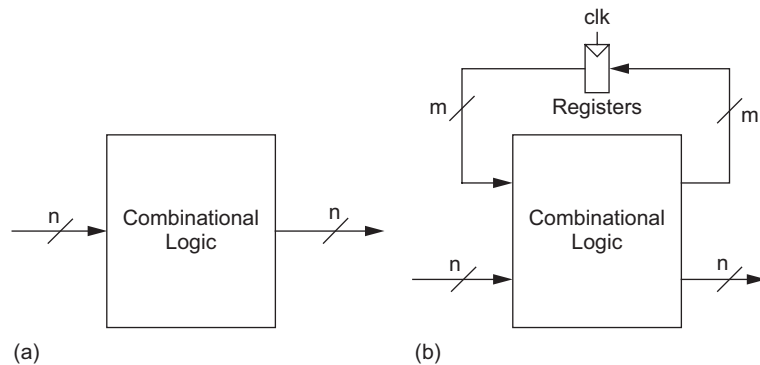


FIGURE 15.6 The combinational explosion in test vectors

Clearly, exhaustive testing is infeasible for most systems. Fortunately, the number of potentially nonfunctional nodes on a chip is much smaller than the number of states. The verification engineer must cleverly devise test vectors that detect any (or nearly any) defective node without requiring so many patterns.

15.3.1 Test Vectors

Test vectors are a set of patterns applied to inputs and a set of expected outputs. Both logic verification and manufacturing test require a good set of test vectors. The set should be

large enough to catch all the logic errors and manufacturing defects, yet small enough to keep test time (and cost) reasonable.

Directed and *random* vectors are the most common types. Directed vectors are selected by an engineer who is knowledgeable about the system. Their purpose is to cover the corner cases where the system might be most likely to malfunction. For example, in a 32-bit datapath, likely corner cases include the following:

0x00000000	All zeros
0xFFFFFFFF	All ones
0x00000001	One in the lsb
0x80000000	One in the msb
0x55555555	Alternating 0's and 1's
0xAAAAAAAA	Alternating 1's and 0's
0x7A39D281	A random value

The circuit could be tested by applying all combinations of these directed vectors to the various inputs. Directed vectors are an efficient way to catch the most obvious design errors and a good logic designer will always run a set of directed tests on a new piece of RTL to ensure a minimum level of quality.

Applying a large number of random or semirandom vectors is a surprisingly good way to detect more subtle errors. The effectiveness of the set of vectors is measured by the fault coverage, which is discussed in Section 15.5.6. Automatic test pattern generation tools are good at producing high fault coverage for manufacturing test and are discussed in Section 15.5.7.

15.3.2 Testbenches and Harnesses

A verification *test bench* or *harness* is a piece of HDL code that is placed as a wrapper around a core piece of HDL to apply and check test vectors. In the simplest test bench, input vectors are applied to the module under test and at each cycle, the outputs are examined to determine whether they comply with a predefined expected data set. The expected outputs can be derived from the golden model and saved as a file or the value can be computed on the fly.

Simulators usually provide settable break points and single or multiple stepping abilities to allow the designer to step through a test sequence while debugging discrepancies.

15.3.3 Regression Testing

High-level language scripts are frequently used when running large testbenches, especially for *regression testing*. Regression testing involves performing a suite of simulations to automatically verify that no functionality has inadvertently changed in a module or set of modules. During a design, it is common practice to run a regression script every night after design activities have concluded to check that bug fixes or feature enhancements have not broken completed modules.

Example 15.3

Figure 14.11 showed a possible software radio architecture that used a combination of an IQ conversion block and a multiplier-based multiprocessor. The following regression testing might be done:

```

Test IQ Conversion
  Test Upconverter
    Test NCO
      Test Read and Write of All Registers
      Test Phase Incrementer
      Test Phase Adder
      Test Sine ROM (Read Contents)
      Test Overall NCO at a set of frequencies
    Test Multiplier
  Test Downconverter
    Test NCO
    ...
    Test Multiplier
    ...
    Test Low Pass Filter
    ...
  Test Microprocessor Memory Core
    Test Microprocessor
      Test ALU
      Test Instruction Decode
      Test Program Counter
      Test Register File Read/Write
      Exhaustive Instruction Test
    Test Memory Read/Write
  Test Interprocessor Bus IO
  Test IQ Conversion to Processor pathways
  Test Overall Software Radio Functionality

```

Note the way in which the correctness of modules is slowly built up by verifying lower-level models first. The low-level tests are gradually built up in complexity until the complete functionality can be verified. At low levels, it is easier to exhaustively verify that logic is correct. For instance, we can verify that the sine ROM is in fact generating a sine wave for one frequency. We then use this knowledge to postulate that it generates correct sine waves for all input frequencies when we verify at the levels above the NCO. At the chip level, we assume that IQ conversion is correct for all combinations of signal frequency and local oscillator frequency even though we may only check a small subset. If we started at the top level and ran a simulation for a few frequencies, we could never have confidence that the lower levels were correct. In addition, if there is a problem, trying to locate the problem by debugging at the top level is futile. Running regression tests from the bottom up is designed to overcome this verification nightmare.

15.3.4 Version Control

Combined with regression testing is the use of versioning, that is, the orderly management of different design iterations. Unix/Linux tools such as CVS or Subversion are useful for this.

Example 15.4

In the software radio example, the regression testing halts at the ALU test in the example given above. Working late, the design leader, Vanessa Eagleeye, examines the CVS

history and discovers that Fred Codechanger has made an edit to the ALU design to try a new adder during the day. She is able to revert the code to what was previously working and then rerun the regression test and have a peaceful night's sleep. Fred corrects his mistake the next day and is advised to remember to run the regression verification step before submitting such hurried edits.

15.3.5 Bug Tracking

Another important tool to use during verification (and in fact the whole design cycle) is a bug-tracking system. Bug-tracking systems such as the Unix/Linux based GNATS allow the management of a wide variety of bugs. In these systems, each bug is entered and the location, nature, and severity of the bug noted. The bug discoverer is noted, along with the perceived person responsible for fixing the bug.

Example 15.5

After Example 15.4, Vanessa enters a bug report describing the bug. She cites Fred as the person responsible and the level as severe. The next day, Fred fixes the problem and changes the bug status to fixed. The bug report is kept in the system, but does not appear in any listing of outstanding bugs. It is kept to track the re-introduction of bugs, as this might give managers an idea of a problem area in the design management.

Tracking the number of bugs can give you an idea of the rate at which a design is converging toward a finished state. If the trend is downward, the design is converging. On the other hand, an upward trend tends to indicate a design early in its verification cycle.

15.4 Silicon Debug Principles

The area of basic digital debugging was introduced in Section 15.1.2. A major challenge in silicon debugging is when the chip operates incorrectly, but you cannot ascertain the cause by making measurements at the chip pins or scan chain outputs (see Section 15.6.2).

There are a number of techniques for directly accessing the silicon. First, specific signals can be brought to the top of the chip as *probe points*. These are small squares (5–10 μm on a side) of top-level metal that connect to key points in the circuit that the designer has had the foresight to include before debug. The overglass cut mask should specify a hole in the passivation over the probe pads so the metal can be reliably contacted. Typical of these kinds of test points might be internal bias points in linear circuits or perhaps key points in a high-speed signal chain (be careful not to excessively load the circuit to be probed). The exposed squares can be probed with a picoprobe (fine-tipped probe) in a fixture under a microscope. During design, the load of the picoprobe has to be taken into account by providing buffers if necessary. The Model 35 probe from GGB Industries shown in Figure 15.7 has a capacitance of 50 fF, input resistance of 1.25 M Ω , and frequency response from DC to 26 GHz. It can probe down to a $10 \times 10 \mu\text{m}$ window.

The die can also be probed electrically or optically if mechanical contact is not feasible. An *electron beam* (ebeam) probe uses a scanning electron microscope to produce a



FIGURE 15.7 GGB Industries Model 15 picoprobe (© 2009 GGB Industries, reprinted with permission.)

tightly focused beam of electrons to measure on-chip voltages. Similarly, *Laser Voltage Probing* (LVP) [Lasserre99] involves shining a laser at a circuit and observing the reflected light. The reflections are modulated by the electric fields so switching waveforms can be deduced. However, the probing can be invasive; the stream of photons may disturb sensitive dynamic nodes. *Picosecond Imaging Circuit Analysis* (PICA) [Knebel98] captures faint light emission naturally produced by switching transistors and hence is noninvasive. Silicon is partially transparent to infrared light, so both LVP and PICA can be performed through the substrate from the backside of a chip in a flip-chip package.

On a more coarse scale, infrared (IR) imaging can be used to examine “hot spots” in a chip, which may be the source of problems (for instance, a resistive short between power rails). There are also liquid crystal materials, which can be “painted on” to a die to indicate temperature problems at a coarse resolution.

If the location of the fault is known, a *Focused Ion Beam* (FIB) can be used to cut wires or lay new conductors down. Even with plastic-packaged parts, the plastic can be carefully ground off and these repairs completed. The reason for this kind of tool is that normally in any chip project, time is of the essence and FIB runs are quicker (and cheaper for a few parts) than frequent mask changes. Laser cutting is also possible. Commercial providers such as MEFAS offer these services.

Example 15.6

A short between V_{DD} and GND has rendered a chip just back from tapeout nonfunctional. The position of the fault is known and it can be corrected by a cut to the top level metal. Several packaged parts are sent to the FIB house with a location from a given fiducial mark and an accompanying plot of the position of the metal to be cut. The FIB house exposes the die (i.e., by grinding a plastic package). The operator then locates the cut position manually using a microscope and runs the FIB machine. The modified packages are then returned to the designers, where hopefully they celebrate the successful test of an otherwise useless chip.

Debugging logic circuits will often involve extremely fast or novel circuits that are largely analog in nature. In this case, it is advisable to have a model of the circuit in question available in SPICE. Debugging analog circuits, as with purely digital circuits, involves making an assertion and then trying to prove the assertion is correct. This can begin with a SPICE simulation and then progress to silicon measurement.

Failures causes may be *manufacturing*, *functional*, or *electrical*. Manufacturing failures occur when a chip has a defect or is outside of the parametric specifications. Debug can reject chips with manufacturing problems, although circuits sensitive to weaknesses in the manufacturing process can be changed to improve yield, as will be discussed in Section 15.6.5. Functional failures are logic bugs or physical design errors that cause the chip to fail under all conditions. They arise from inadequate logic verification and are usually the easiest to fix. Electrical failures occur when the chip is logically correct, but malfunctions under certain conditions such as voltage, temperature, or frequency. Section 9.3 addressed many causes of electrical failures. Some electrical failures can be so severe that they appear as functional failures, while others occur rarely and are extremely difficult to reproduce and diagnose.

So-called shmoo plots can help to debug electrical failures in silicon [Baker97]. A shmoo plot is often made with voltage on one axis and speed on the other. The test vectors are applied at each combination of voltage and clock speed, and the success of the test is recorded. Often, only a set of vectors applicable to a particular module is applied to diagnose a problem in that module.

Figure 15.8 shows a shmoo from the Intel Atom microprocessor [Gerosa09]. Dots in the light gray area indicate correct operation, while different letters indicate different failure modes. The chip works at 1.25 GHz and 0.75 V and at 2.5 GHz at 1.15 V.

The shmoo plots shown in Figure 15.9 illustrate a variety of conditions [Josephson02]. A healthy normal chip should operate at increasing frequency as the voltage increases. The brick wall pattern suggests that the chip may be randomly initialized in one of two states, only one of which is correct. For example, a register without a reset signal may randomly have an initial state of 0 or 1. The wall pattern in which the chip fails to operate at any frequency above or below a particular voltage can indicate charge sharing, coupling noise, or a race condition. The reverse speedpath behavior indicates a leakage problem in which a weakly held node leaks to an invalid level before the end of the cycle. At higher voltage, the leakage is exacerbated and appears at shorter clock periods. The floor is a variant on the leakage problem where the part fails at low frequency independent of the voltage. A finger indicates coupling problems dependent on the alignment of the aggressor and victim, where at certain frequencies the alignment always causes a failure.

A shmoo can also plot operating speed against temperature. At cold temperature, FETs are faster, have lower effective resistance, and have higher threshold voltages. A normal shmoo should show speed increasing as temperature decreases. Failures at low temperature could indicate coupling or charge sharing noise exacerbated by faster edge rates. Failures at high temperature could indicate excessive leakage or noise problems exacerbated by the lower threshold voltages. Walls at either temperature could indicate race conditions where the path that wins the race varies with temperature.

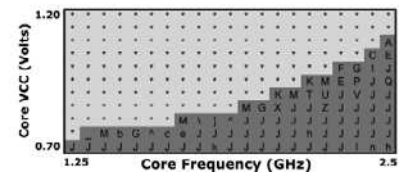
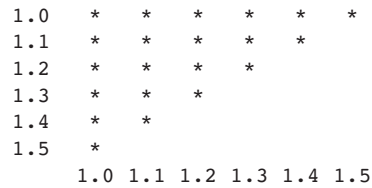
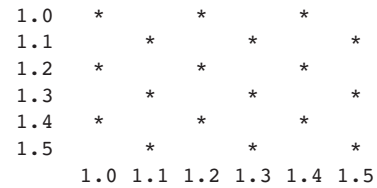


FIGURE 15.8 Shmoo for Intel Atom microprocessor (© IEEE 2009.)

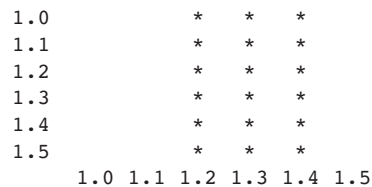
Clock period in ns on the left, frequency increases going up
Voltage on the bottom, increase left to right
* indicates a failure



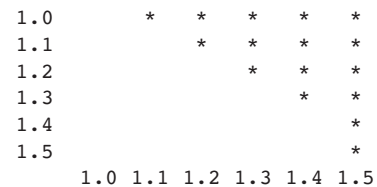
Normal
Well-behaved shmoo
Typical speedpath



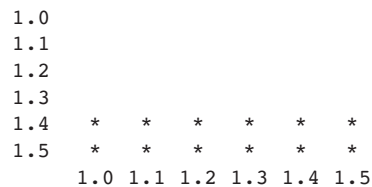
“Brick Wall”
Bistable
Initialization



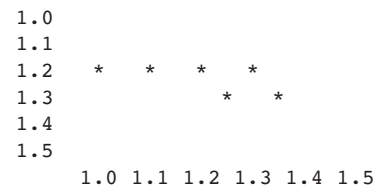
“Wall”
Fails at a certain voltage
Coupling, charge share, races



“Reverse Speedpath”
Increase in voltage reduces frequency
Speedpath, leakage



“Floor”
Works at high but not low frequency
Leakage



“Finger”
Fails at a specific point in the shmoo
Coupling

FIGURE 15.9 Shmoo plots with symptoms

15.5 Manufacturing Test Principles

As discussed in Section 14.5.2, integrated circuits have a yield of less than 100%. Figure 15.10 shows micrographs of some manufacturing defects.

The purpose of manufacturing test is to screen out most of the defective parts before they are shipped to customers. Typical commercial products target a defect rate of 350–1000 defects per million (DPM) chips shipped. The customer then assembles

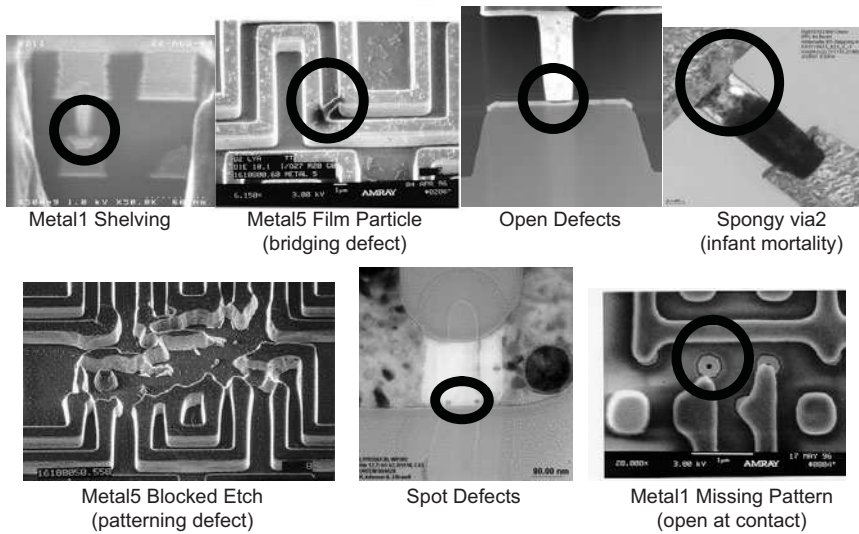


FIGURE 15.10 SEM images of manufacturing defects (Courtesy of Intel Corporation.)

systems from the chips, tests the systems, and discards or repairs defective systems. A high defect rate leads to unhappy customers.

A critical factor in all VLSI design is the need to incorporate methods of testing circuits. This task should proceed concurrently with architectural considerations and not be left until fabricated parts are available (as is a recurring temptation to designers).

15.5.1 Fault Models

To deal with the existence of good and bad parts, it is necessary to propose a *fault model*; i.e., a model for how faults occur and their impact on circuits. The most popular model is called the *Stuck-At* model. The *Short Circuit/Open Circuit* model can be a closer fit to reality, but is harder to incorporate into logic simulation tools.

15.5.1.1 Stuck-At Faults In the Stuck-At model, a faulty gate input is modeled as a *stuck at zero* (Stuck-At-0, S-A-0) or *stuck at one* (Stuck-At-1, S-A-1). This model dates from board-level designs, where it was determined to be adequate for modeling faults. Figure 15.11 illustrates how an S-A-0 or S-A-1 fault might occur. These faults most frequently occur due to gate oxide shorts (the nMOS gate to GND or the pMOS gate to V_{DD}) or metal-to-metal shorts.

15.5.1.2 Short-Circuit and Open-Circuit Faults Other models include *stuck-open* or *shorted* models [Jayasumana91]. Two bridging or shorted faults are shown in Figure 15.12. The short $S1$ results in an S-A-0

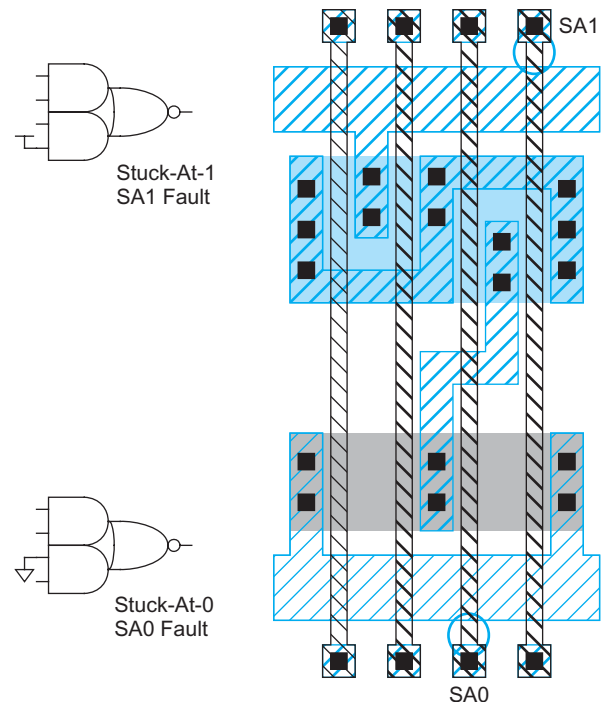


FIGURE 15.11 CMOS stuck-at faults

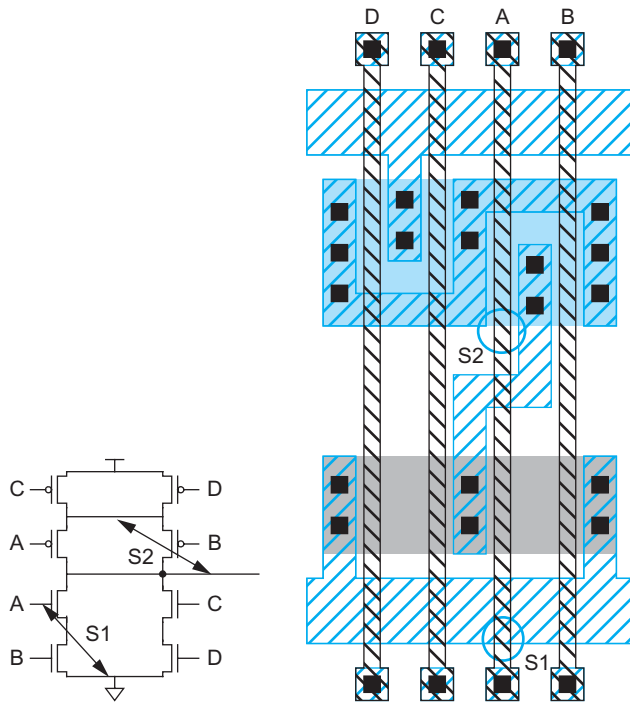


FIGURE 15.12 CMOS bridging faults

fault at input A, while short *S2* modifies the function of the gate. It is evident that to ensure the most accurate modeling, faults should be modeled at the transistor level because it is only at this level that the complete circuit structure is known. For instance, in the case of a simple NAND gate, the intermediate node between the series nMOS transistors is hidden by the schematic. This implies that test generation should ideally take account of possible shorts and open circuits at the switch level [Galiay80]. Expediency dictates that most existing systems rely on Boolean logic representations of circuits and stuck-at fault modeling.

A particular problem that arises with CMOS is that it is possible for a fault to convert a combinational circuit into a sequential circuit. This is illustrated in Figure 15.13 for the case of a 2-input NOR gate in which one of the transistors is rendered ineffective. If nMOS transistor *A* is stuck open, then the function displayed by the gate will be

$$Z = \overline{A + B} + \overline{B}Z' \quad (15.1)$$

where Z' is the previous state of the gate. As another example, if either pMOS transistor is missing, the node would be arbitrarily charged (i.e., it might be high due to some weird charging sequence) until one of

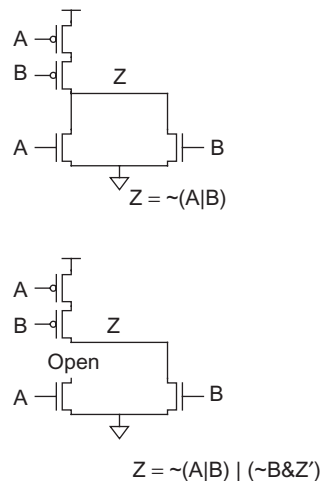
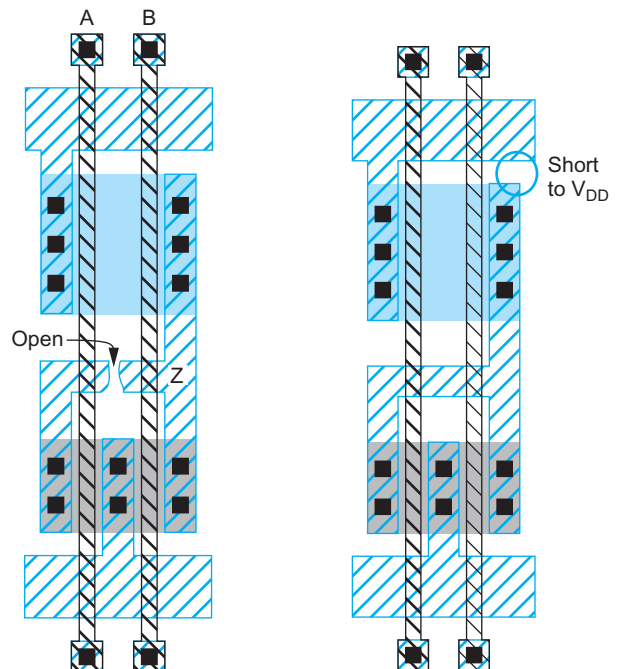


FIGURE 15.13 A CMOS open fault that causes sequential faults

FIGURE 15.14 A defect that causes static I_{DD} current

the nMOS transistors discharged the node. Thereafter, it would remain at zero, barring charge leakage effects.

It is also possible for transistors to exhibit a stuck-open or stuck-closed state. Stuck-closed states can be detected by observing the static V_{DD} current (I_{DD}) while applying test vectors. Consider the fault shown in Figure 15.14, where the drain connection on a pMOS transistor in a 2-input NOR gate is shorted to V_{DD} . This could physically occur if stray metal (caused by a speck of dust at the photolithography stage) overlapped the V_{DD} line and drain connection as shown. If we apply the test vector 01 or 10 to the A and B inputs and measure the static I_{DD} current, we will notice that it rises to some value determined by size of the nMOS transistors.

15.5.2 Observability

The *observability* of a particular circuit node is the degree to which you can observe that node at the outputs of an integrated circuit (i.e., the pins). This metric is relevant when you want to measure the output of a gate within a larger circuit to check that it operates correctly. Given the limited number of nodes that can be directly observed, it is the aim of good chip designers to have easily observed gate outputs. Adoption of some basic design for test techniques can aid tremendously in this respect. Ideally, you should be able to observe directly or with moderate indirection (i.e., you may have to wait a few cycles) every gate output within an integrated circuit. While at one time this aim was hindered by the expense of extra test circuitry and a lack of design methodology, current processes and design practices allow you to approach this ideal. Section 15.6 examines a range of methods for increasing observability.

15.5.3 Controllability

The *controllability* of an internal circuit node within a chip is a measure of the ease of setting the node to a 1 or 0 state. This metric is of importance when assessing the degree of difficulty of testing a particular signal within a circuit. An easily controllable node would be directly settable via an input pad. A node with little controllability, such as the most significant bit of a counter, might require many hundreds or thousands of cycles to get it to the right state. Often, you will find it impossible to generate a test sequence to set a number of poorly controllable nodes into the right state. It should be the aim of good chip designers to make all nodes easily controllable. In common with observability, the adoption of some simple design for test techniques can aid in this respect tremendously. Making all flip-flops resettable via a global reset signal is one step toward good controllability.

15.5.4 Repeatability

The *repeatability* of system is the ability to produce the same outputs given the same inputs. Combinational logic and synchronous sequential logic is always repeatable when it is functioning correctly. However, certain asynchronous sequential circuits are nondeterministic. For example, an arbiter may select either input when both arrive at nearly the same time. Testing is much easier when the system is repeatable. Some systems with asynchronous interfaces have a lock-step mode to facilitate repeatable testing.

15.5.5 Survivability

The *survivability* of a system is the ability to continue function after a fault. For example, error-correcting codes provide survivability in the event of soft errors. Redundant rows and columns in memories and spare cores provide survivability in the event of manufactur-

ing defects. Adaptive techniques provide survivability in the event of process variation. Some survivability features are invoked automatically by the hardware, while others are activated by blowing fuses after manufacturing test.

15.5.6 Fault Coverage

A measure of goodness of a set of test vectors is the amount of *fault coverage* it achieves. That is, for the vectors applied, what percentage of the chip's internal nodes were checked? Conceptually, the way in which the fault coverage is calculated is as follows. Each circuit node is taken in sequence and held to 0 (S-A-0), and the circuit is simulated with the test vectors comparing the chip outputs with a *known good machine*—a circuit with no nodes artificially set to 0 (or 1). When a discrepancy is detected between the *faulty machine* and the good machine, the fault is marked as detected and the simulation is stopped. This is repeated for setting the node to 1 (S-A-1). In turn, every node is stuck (artificially) at 1 and 0 sequentially. The fault coverage of a set of test vectors is the percentage of the total nodes that can be detected as faulty when the vectors are applied. To achieve world-class quality levels, circuits are required to have in excess of 98.5% fault coverage. The *Verification Methodology Manual* [Bergeron05] is the bible for fault coverage techniques.

15.5.7 Automatic Test Pattern Generation (ATPG)

Historically, in the IC industry, logic and circuit designers implemented the functions at the RTL or schematic level, mask designers completed the layout, and test engineers wrote the tests. In many ways, the test engineers were the Sherlock Holmes of the industry, reverse engineering circuits and devising tests that would test the circuits in an adequate manner. For the longest time, test engineers implored circuit designers to include extra circuitry to ease the burden of test generation. Happily, as processes have increased in density and chips have increased in complexity, the inclusion of test circuitry has become less of an overhead for both the designer and the manager worried about the cost of the die. In addition, as tools have improved, more of the burden for generating tests has fallen on the designer. To deal

with this burden, *Automatic Test Pattern Generation* (ATPG) methods have been invented. The use of some form of ATPG is standard for most digital designs.

Commercial ATPG tools can achieve excellent fault coverage. However, they are computation-intensive and often must be run on servers or compute farms with many parallel processors. Some tools use statistical algorithms to predict the fault coverage of a set of vectors without performing as much simulation. Adding scan and built-in self-test, as described in Section 15.6, improves the observability of a system and can reduce the number of test vectors required to achieve a desired fault coverage.

15.5.8 Delay Fault Testing

The fault models dealt with until this point have neglected timing. Failures that occur in CMOS could leave the functionality of the circuit untouched, but affect the timing. For

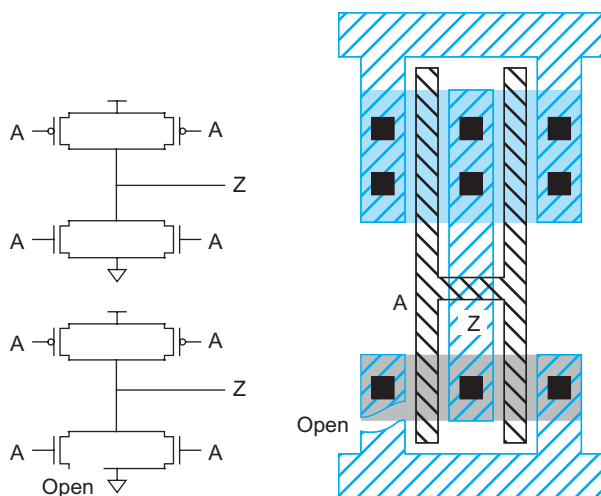


FIGURE 15.15 An example of a delay fault

instance, consider the layout shown in Figure 15.15 for an inverter gate composed of paralleled nMOS and pMOS transistors. If an open circuit occurs in one of the nMOS transistor source connections to GND, then the gate would still function but with increased t_{pdf} . In addition, the fault now becomes sequential as the detection of the fault depends on the previous state of the gate.

Delay faults may be caused by crosstalk [Paul02]. Delay faults can also occur more often in SOI logic through the history effect. Software has been developed to model the effect of delay faults and is becoming more important as a failure mode as processes scale.

15.6 Design for Testability

The keys to designing circuits that are testable are controllability and observability. Restated, controllability is the ability to set (to 1) and reset (to 0) every node internal to the circuit. Observability is the ability to observe, either directly or indirectly, the state of any node in the circuit. Good observability and controllability reduce the cost of manufacturing testing because they allow high fault coverage with relatively few test vectors. Moreover, they can be essential to silicon debug because physically probing internal signals has become so difficult.

We will first cover three main approaches to what is commonly called *Design for Testability* (DFT). These may be categorized as follows:

- *Ad hoc* testing
- Scan-based approaches
- Built-in self-test (BIST)

15.6.1 *Ad Hoc* Testing

Ad hoc test techniques, as their name suggests, are collections of ideas aimed at reducing the combinational explosion of testing. They are summarized here for historical reasons. They are only useful for small designs where scan, ATPG, and BIST are not available. A complete scan-based testing methodology is recommended for all digital circuits. Having said that, the following are common techniques for *ad hoc* testing:

- Partitioning large sequential circuits
- Adding test points
- Adding multiplexers
- Providing for easy state reset

A technique classified in this category is the use of the bus in a bus-oriented system for test purposes. Each register has been made loadable from the bus and capable of being driven onto the bus. Here, the internal logic values that exist on a data bus are enabled onto the bus for testing purposes.

Frequently, multiplexers can be used to provide alternative signal paths during testing. In CMOS, transmission gate multiplexers provide low area and delay overhead.

Any design should always have a method of resetting the internal state of the chip within a single cycle or at most a few cycles. Apart from making testing easier, this also makes simulation faster as a few cycles are required to initialize the chip.

In general, *ad hoc* testing techniques represent a bag of tricks developed over the years by designers to avoid the overhead of a systematic approach to testing, as will be described in the next section. While these general approaches are still quite valid, process densities and chip complexities necessitate a structured approach to testing.

15.6.2 Scan Design

The *scan-design* strategy for testing has evolved to provide observability and controllability at each register. In designs with scan, the registers operate in one of two modes. In *normal mode*, they behave as expected. In *scan mode*, they are connected to form a giant shift register called a *scan chain* spanning the whole chip. By applying N clock pulses in scan mode, all N bits of state in the system can be shifted out and new N bits of state can be shifted in. Therefore, scan mode gives easy observability and controllability of every register in the system.

Modern scan is based on the use of scan registers, as shown in Figure 15.16. The scan register is a D flip-flop preceded by a multiplexer. When the *SCAN* signal is deasserted, the register behaves as a conventional register, storing data on the D input. When *SCAN* is asserted, the data is loaded from the *SI* pin, which is connected in shift register fashion to the previous register Q output in the scan chain.

For the circuit to load the scan chain, *SCAN* is asserted and *CLK* is pulsed eight times to load the first two ranks of 4-bit registers with data. *SCAN* is deasserted and *CLK* is asserted for one cycle to operate the circuit normally with predefined inputs. *SCAN* is then reasserted and *CLK* asserted eight times to read the stored data out. At the same time, the

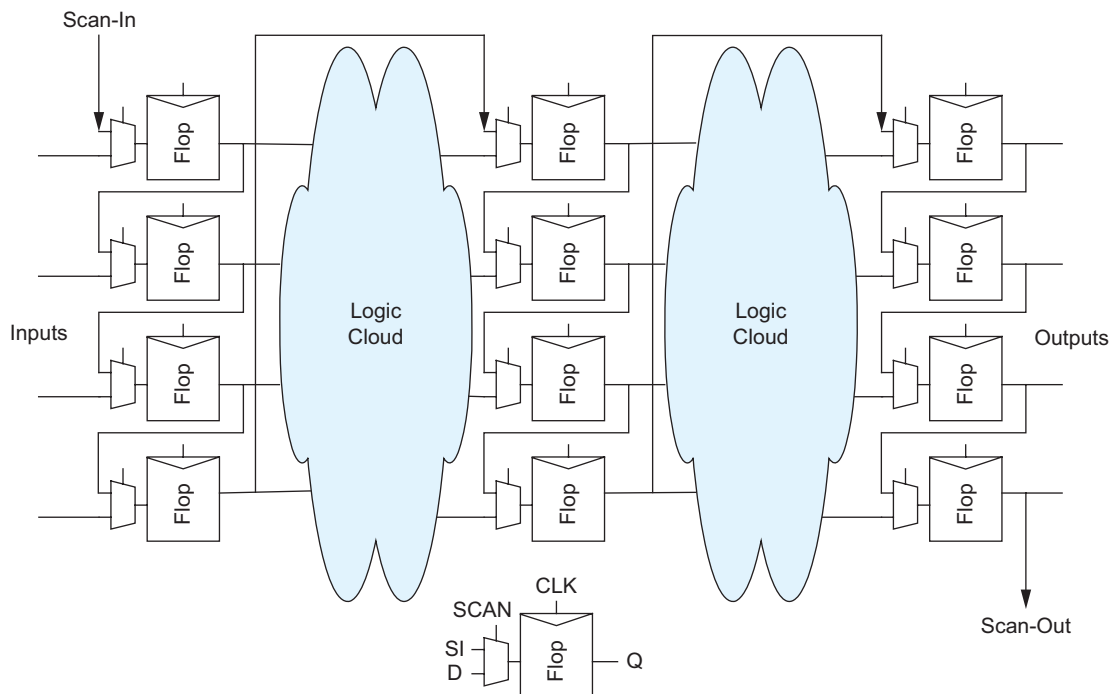


FIGURE 15.16 Scan-based testing

new register contents can be shifted in for the next test. Testing proceeds in this manner of serially clocking the data through the scan register to the right point in the circuit, running a single system clock cycle and serially clocking the data out for observation. In this scheme, every input to the combinational block can be controlled and every output can be observed. In addition, running a random pattern of 1s and 0s through the scan chain can test the chain itself.

Test generation for this type of test architecture can be highly automated. ATPG techniques can be used for the combinational blocks and, as mentioned, the scan chain is easily tested. The prime disadvantage is the area and delay impact of the extra multiplexer in the scan register. Designers (and managers alike) are in widespread agreement that this cost is more than offset by the savings in debug time and production test cost.

15.6.2.1 Parallel Scan You can imagine that serial scan chains can become quite long, and the loading and unloading can dominate testing time. A fairly simple idea is to split the chains into smaller segments. This can be done on a module-by-module basis or completed automatically to some specified scan length. Extending this to the limit yields an extension to serial scan called *random access scan* [Ando80]. To some extent, this is similar to that used inside FPGAs to load and read the control RAM.

The basic idea is shown in Figure 15.17. The figure shows a two-by-two register section. Each register receives a column (`column<m>`) and row (`row<n>`) access signal along with a row data line (`data<n>`). A global write signal (`write`) is connected to all registers. By asserting the row and column access signals in conjunction with the write signal, any register can be read or written in exactly the same method as a conventional RAM. The

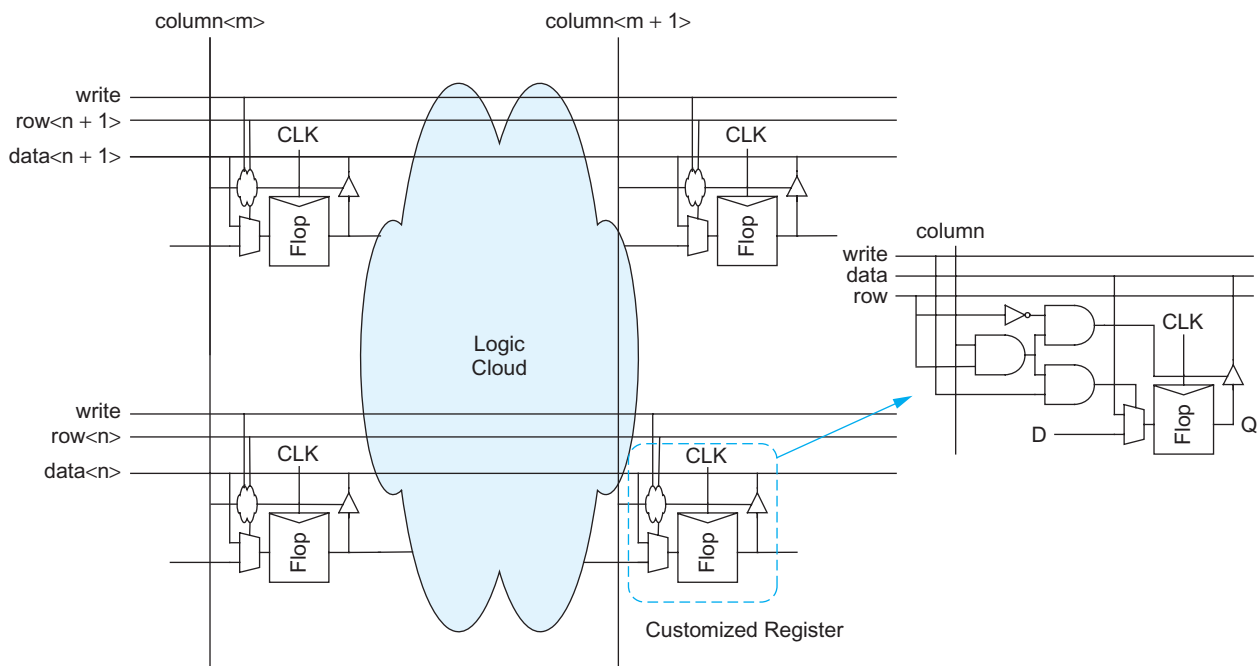


FIGURE 15.17 Parallel scan—basic structure

notional logic is shown to the right of the four registers. Implementing the logic required at the transistor level can reduce the overhead for each register.

15.6.2.2 Scannable Register Design As we have seen, an ordinary flip-flop can be made scannable by adding a multiplexer on the data input, as shown in Figure 15.18(a). Figure 15.18(b) shows a circuit design for such a scan register using a transmission-gate multiplexer. The setup time increases by the delay of the extra transmission gate in series with the D input as compared to the ordinary static flip-flop shown in Figure 10.19(b). Figure 15.18(c) shows a circuit using clock gating to obtain nearly the same setup time as the ordinary flip-flop. In either design, if a clock enable is used to stop the clock to unused portions of the chip, care must be taken that ϕ always toggles during scan mode.

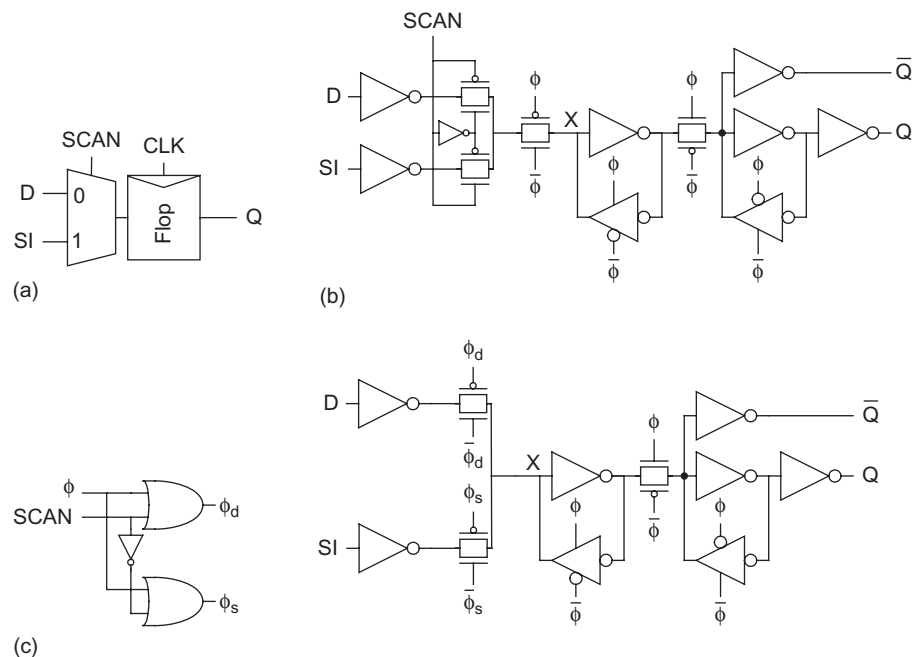


FIGURE 15.18 Scannable flip-flops



15.6.2.3 Other Scannable Elements

This section is available in the online Web Enhanced chapter at www.cmosvlsi.com.

15.6.3 Built-In Self-Test (BIST)

Self-test and built-in test techniques, as their names suggest, rely on augmenting circuits to allow them to perform operations upon themselves that prove correct operation. These techniques add area to the chip for the test logic, but reduce the test time required and thus can lower the overall system cost. [Stroud02] offers extensive coverage of the subject from the implementer's perspective.

One method of testing a module is to use *signature analysis* [Frowerk77, Nadig77] or *cyclic redundancy checking*. This involves using a *pseudo-random sequence generator* (PRSG)

to produce the input signals for a section of combinational circuitry and a *signature analyzer* to observe the output signals.

A PRSG of length n is constructed from a *linear feedback shift register* (LFSR), which in turn is made of n flip-flops connected in a serial fashion, as shown in Figure 15.19(a). The XOR of particular outputs are fed back to the input of the LFSR. An n -bit LFSR will cycle through $2^n - 1$ states before repeating the sequence. LFSRs are discussed further in Section 11.5.4. They are described by a *characteristic polynomial* indicating which bits are fed back. A *complete feedback shift register* (CFSR), shown in Figure 15.19(b), includes the zero state that may be required in some test situations [Wang86]. An n -bit LFSR is converted to an n -bit CFSR by adding an $n - 1$ input NOR gate connected to all but the last bit. When in state $0 \dots 01$, the next state is $0 \dots 00$. When in state $0 \dots 00$, the next state is $10 \dots 0$. Otherwise, the sequence is the same. Alternatively, the bottom n bits of an $n + 1$ -bit LFSR can be used to cycle through the all zeros state without the delay of the NOR gate.

A signature analyzer receives successive outputs of a combinational logic block and produces a *syndrome* that is a function of these outputs. The syndrome is reset to 0, and then XORed with the output on each cycle. The syndrome is swizzled each cycle so that a fault in one bit is unlikely to cancel itself out. At the end of a test sequence, the LFSR contains the syndrome that is a function of all previous outputs. This can be compared with the correct syndrome (derived by running a test program on the good logic) to determine whether the circuit is good or bad. If the syndrome contains enough bits, it is improbable that a defective circuit will produce the correct syndrome.

15.6.3.1 BIST The combination of signature analysis and the scan technique creates a structure known as *BIST*—for *Built-In Self-Test* or *BILBO*—for *Built-In Logic Block Observation* [Koenemann79]. The 3-bit BIST register shown in Figure 15.20 is a scanable, resettable register that also can serve as a pattern generator and signature analyzer. $C[1:0]$ specifies the mode of operation. In the reset mode (10), all the flip-flops are synchronously initialized to 0. In normal mode (11), the flip-flops behave normally with their D input and Q output. In scan mode (00), the flip-flops are configured as a 3-bit shift register between SI and SO . Note that there is an inversion between each stage. In test mode (01), the register behaves as a pseudo-random sequence generator or signature analyzer. If all the D inputs are held low, the Q outputs loop through a pseudo-random bit sequence, which can serve as the input to the combinational logic. If the D inputs are taken from the combinational logic output, they are swizzled with the existing state to produce the syndrome. In summary, BIST is performed by first resetting the syndrome in the output register. Then both registers are placed in the test mode to produce the pseudo-random inputs and calculate the syndrome. Finally, the syndrome is shifted out through the scan chain.

Various companies have commercial design aid packages that automatically replace ordinary registers with scannable BIST registers, check the fault coverage, and generate scripts for production testing. As an example, on a WLAN modem chip comprising roughly 1 million gates, a full at-speed test takes under a second with BIST. This comes with roughly a 7.3% overhead in the core area (but actually zero because the design was

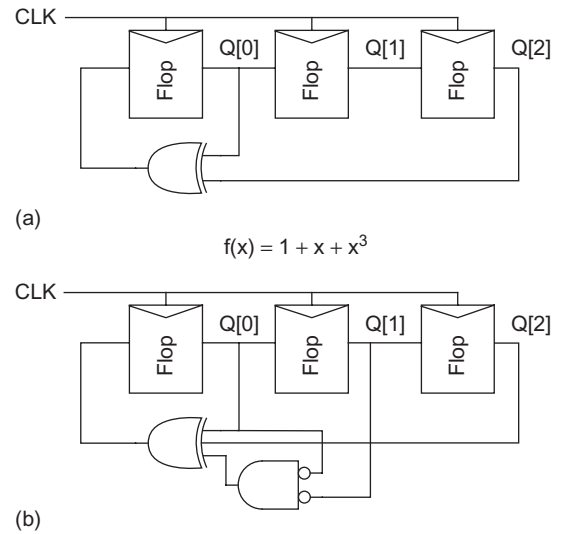


FIGURE 15.19 Pseudo-random sequence generator

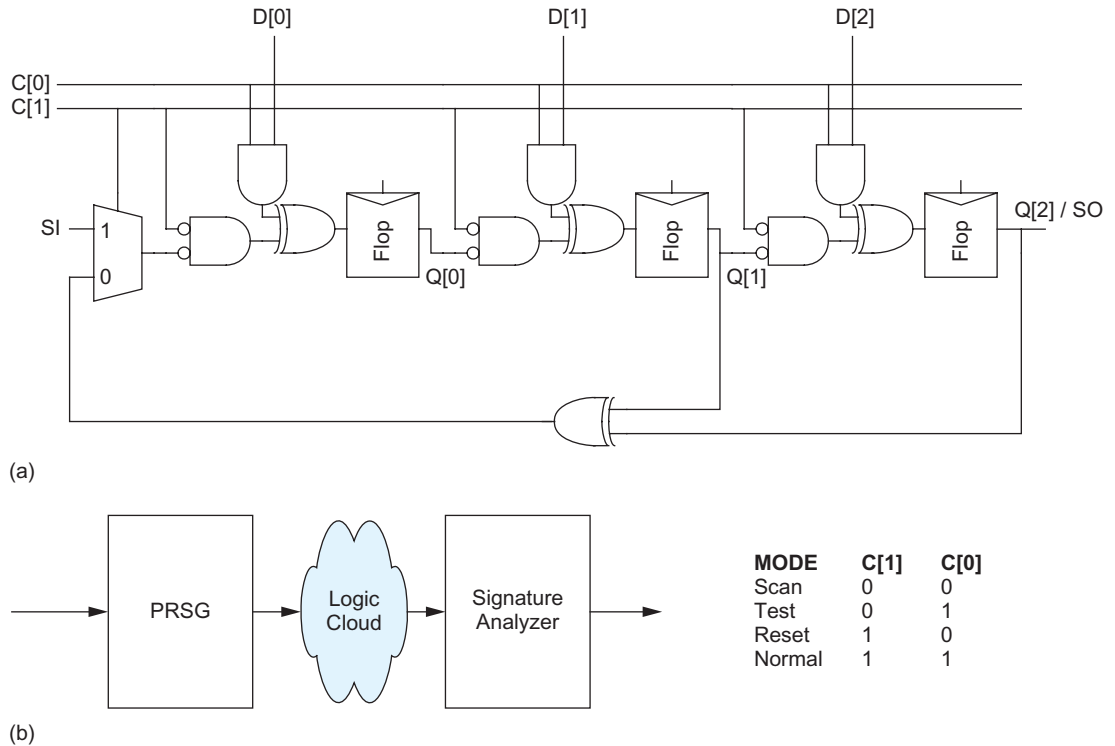


FIGURE 15.20 BIST (a) 3-bit register, (b) use in a system

pad limited) and a 99.7% fault coverage level. The WLAN modem parts designed in this way were fully tested in less than ten minutes on receipt of first silicon. This kind of test method is incredibly valuable for productivity in manufacturing test generation.

15.6.3.2 Memory BIST On many chips, memories account for the majority of the transistors. A robust testing methodology must be applied to provide reliable parts. In a typical MBIST scheme, multiplexers are placed on the address, data, and control inputs for the memory to allow direct access during test. During testing, a state machine uses these multiplexers to directly write a checkerboard pattern of alternating 1s and 0s. The data is read back, checked, then the inverse pattern is also applied and checked. ROM testing is even simpler: The contents are read out to a signature analyzer to produce a syndrome.

15.6.3.3 Other On-Chip Test Strategies On-chip speeds are usually so high that directly observing internal behavior for testing can be difficult or impossible. Designers have included on-chip logic analyzers and oscilloscopes to deal with this problem [Weinlader00, Lee06, Noguchi07]. Such systems typically require a trigger signal to initiate data collection, a high speed timing generator, analog or digital sampling, and a buffer to store the results until they can be off-loaded at lower speed. A drawback is that the nodes to be observed must be selected at design time, and these may not be the problem circuits. Nevertheless, probing major busses and critical analog/RF nodes can be helpful. Also, on-chip scopes have been used to characterize power supply noise [Alon05, Naffziger06] and clock jitter [Nose06].

Analog/digital converter testing requires real-time access to the digital output of the ADC. Providing parallel digital test ports by reassigning pins on the chip I/O can facilitate this testing. If this is impossible, a “capture RAM” on chip can be used to capture results in real-time and then the contents can be transferred off-chip at a slower rate for analysis.

If both ADCs and DACs are present, a loopback strategy can be employed, as shown in Figure 15.21. Both analog and digital signals can loop back. Communication and graphics systems frequently have I/O systems that can be configured as shown. It is often worthwhile to add a DAC and an ADC to a system to allow a level of analog self-test.

Providing on-chip debug circuitry involves quite a bit of imagination and forethought in terms of what might go wrong. It is often called “defensive design.” Today, transistor counts and routing resources make it possible to include very sophisticated debug tools provided thought is given to the matter.

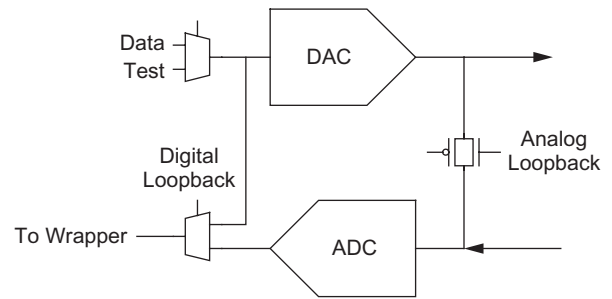


FIGURE 15.21 Analog and digital loopback

15.6.4 IDDQ Testing

Bridging faults were introduced in Section 15.5.1.2. A method of testing for bridging faults is called $IDDQ$ test (V_{DD} supply current Quiescent) or supply current monitoring [Acken83, Lee92]. This relies on the fact that when a CMOS logic gate is not switching, it draws no DC current (except for leakage). When a bridging fault occurs, then for some combination of input conditions, a measurable DC I_{DD} will flow. Testing consists of applying the normal vectors, allowing the signals to settle, and then measuring I_{DD} . As potentially only one gate is affected, the $IDDQ$ test has to be very sensitive. In addition, to be effective, any circuits that draw DC power such as pseudo-nMOS gates or analog circuits have to be disabled. Dynamic gates can also cause problems. As current measuring is slow, the tests must be run slower (of the order of 1 ms per vector) than normal, which increases the test time.

$IDDQ$ testing can be completed externally to the chip by measuring the current drawn on the V_{DD} line or internally using specially constructed test circuits. This technique gives a form of indirect massive observability at little circuit overhead. However, as subthreshold leakage current increases, $IDDQ$ testing ceases to be effective because variations in subthreshold leakage exceed currents caused by the faults.

15.6.5 Design for Manufacturability

Circuits can be optimized for manufacturability to increase their yield. This can be done in a number of different ways.

15.6.5.1 Physical At the physical level (i.e., mask level), the yield and hence manufacturability can be improved by reducing the effect of process defects. The design rules for particular processes will frequently have guidelines for improving yield. The following list is representative:

- Increase the spacing between wires where possible—this reduces the chance of a defect causing a short circuit.

- Increase the overlap of layers around contacts and vias—this reduces the chance that a misalignment will cause an aberration in the contact structure.
- Increase the number of vias at wire intersections beyond one if possible—this reduces the chance of a defect causing an open circuit.

Increasingly, design tools are dealing with these kinds of optimizations automatically.

15.6.5.2 Redundancy Redundant structures can be used to compensate for defective components on a chip. For example, memory arrays are commonly built with extra rows. During manufacturing test, if one of the words is found to be defective, the memory can be reconfigured to access the spare row instead. Laser-cut wires or electrically programmable fuses can be used for configuration. Similarly, if the memory has many banks and one or more are found to be defective, they can be disabled, possibly even under software control.

15.6.5.3 Power Elevated power can cause failure due to excess current in wires, which in turn can cause metal migration failures. In addition, high-power devices raise the die temperature, degrading device performance and, over time, causing device parameter shifts. The method of dealing with this component of manufacturability is to minimize power through design techniques described elsewhere in this text. In addition, a suitable package and heat sink should be chosen to remove excess heat.

15.6.5.4 Process Spread We have seen that process simulations can be carried out at different process corners. Monte Carlo analysis can provide better modeling for process spread and can help with centering a design within the process variations.

15.6.5.5 Yield Analysis When a chip has poor yield or will be manufactured in high volume, dice that fail manufacturing test can be taken to a laboratory for yield analysis to locate the root cause of the failure. If particular structures are determined to have caused many of the failures, the layout of the structures can be redesigned. For example, during volume production ramp-up for the Pentium microprocessor, the silicide over long thin polysilicon lines was found to crack and raise the wire resistance [Needham98]. This in turn led to slower-than-expected operation for the cracked chips. The layout was modified to widen polysilicon wires or strap them with metal wherever possible, boosting the yield at higher frequencies.

15.7 Boundary Scan

Up to this point we have concentrated on the methods of testing individual chips. Many system defects occur at the board level, including open or shorted printed circuit board traces and incomplete solder joints. At the board level, “bed-of-nails” testers historically were used to test boards. In this type of a tester, the board-under-test is lowered onto a set of test points (nails) that probe points of interest on the board. These can be sensed (the observable points) and driven (the controllable points) to test the complete board. At the chassis level, software programs are frequently used to test a complete board set. For instance, when a computer boots, it might run a memory test on the installed memory to detect possible faults.

The increasing complexity of boards and the movement to technologies such as surface mount technologies (with an absence of throughboard vias) resulted in system design-

ers agreeing on a unified scan-based methodology called *boundary scan* for testing chips at the board (and system) level. Boundary scan was originally developed by the Joint Test Access Group and hence is commonly referred to as JTAG. Boundary scan has become a popular standard interface for controlling BIST features as well.

The IEEE 1149 boundary scan architecture [IEEE1149.1-01, Parker03] is shown in Figure 15.22. All of the I/O pins of each IC on the board are connected serially in a standardized scan chain accessed through the *Test Access Port* (TAP) so that every pin can be observed and controlled remotely through the scan chain. At the board level, ICs obeying the standard can be connected in series to form a scan chain spanning the entire board. Connections between ICs are tested by scanning values into the outputs of each chip and checking that those values are received at the inputs of the chips they drive. Moreover, chips with internal scan chains and BIST can access those features through boundary scan to provide a unified testing framework.

Details of boundary scan operation are available in the online Web Enhanced chapter at www.cmosvlsi.com.

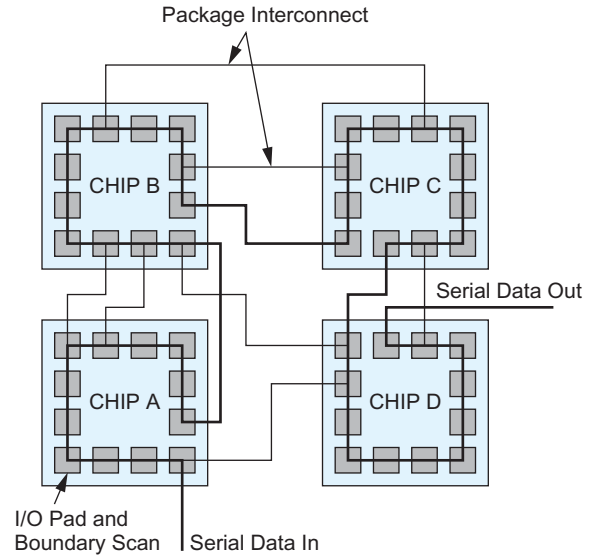


FIGURE 15.22 Boundary scan architecture

15.8 Testing in a University Environment

Industry environments are usually well-funded, and the appropriate testability tools are available to ensure a product-grade test effort. But what do you do in a university environment when the infrastructure might not be quite as affluent as in the industry setting? Not only may test tools be unavailable, but also the very act of building a test board can be a daunting extra amount of work on top of the chip design. The following are some tips that might help in this situation.

Taking the time to include circuitry to aid in testing on the chip is usually much easier than adding it at the board level. For a start, the integrated environment available for most IC design flows allows the designer to simulate the test circuitry. So, while it might seem superfluous to the task at hand, including test circuitry can save a huge amount of effort after the chip returns. Moreover, on-chip circuitry can often test at speeds that are impossible off-chip without extremely expensive production test machines. The main point is to think ahead.

Boundary scan and BIST greatly simplifies testing. If the chip has a standard boundary scan interface, it can be tested from a PC using a commercial boundary scan controller. For example, the Corelis NetUSB-1149.1/E can drive the scan chains at up to 80 MHz.

In the absence of BIST, there are several ways to test a chip. One is to breadboard or wirewrap a test board with switches for inputs and LEDs for outputs. This is tedious for all but the simplest chips. A custom-printed circuit board test fixture is even more labor-intensive, but often necessary for high-performance research chips. Another strategy is to use a logic analyzer with pattern generator. This approach requires a specialized test fixture to hold the chip and often has a steep learning curve for students, but it can perform tests at tens to hundreds of MHz. An increasingly popular method of testing digital chips is to



design a test board that includes a large FPGA. The FPGA can drive test patterns to the chip under test and can store or analyze the responses. Figure 15.23 shows a typical setup.

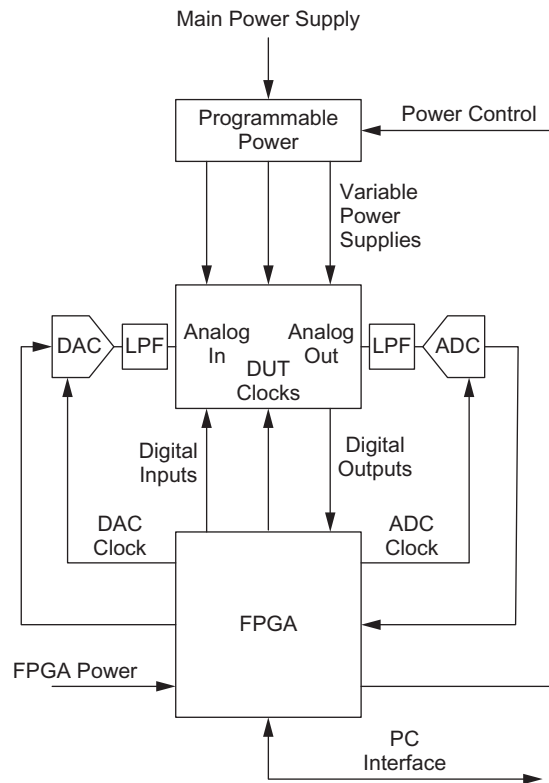


FIGURE 15.23 FPGA-assisted testing

15.9 Pitfalls and Fallacies

The following “war stories” are collected from real products at a wide variety of companies and published with permission, often under the condition of anonymity. They are presented to illustrate some of the pitfalls that can happen to smart people who are dealing with complex systems on a tight schedule. The skilled engineer learns from these mistakes; in most cases, the company extended their verification flow to ensure that similar problems would be caught before wreaking havoc on future products. Could one of these happen to you?

A product in the field hangs unpredictably

A microprocessor had been in the field for several years when reports began arriving from major customers that certain programs would cause the system to hang at unpredictable times with intervals of hours to days. The manufacturer appointed a tiger team to resolve the error. The hang rate proved to be insensitive to power supply voltage, operating temperature, and clock rate. It was observed on all versions of the chip regardless of foundry, manufacturing technology, or motherboard. The programs that failed all involved a mix of floating point and integer operations, not just integer codes.

After several months of work, the issue was isolated to a particular unit in the processor. By this point, 30 engineers were involved in chasing the problem. Picoprobng showed that when the hang occurred, an instruction was left stuck in the pipeline waiting to issue. A logic simulation of the RTL is much slower than running the actual code, but an engineer developed a simple test case that could trigger the hang on real hardware in a matter of seconds, and thus it could trigger the failure in simulation in a practical amount of time. Simulations showed that the RTL ran flawlessly, suggesting the error involved a circuit that did not match the RTL.

On this processor, the circuits had been verified against the RTL using a technique called “shadow-mode simulation.” A “circuit understanding” tool parsed the transistor-level netlist into gates and identified the logic function of each gate. Circuits were verified to match the RTL by replacing a module of the RTL with the corresponding extracted circuit and simulating to check that the system produced identical results as the original RTL. The simulation is time-consuming, so each module is typically checked over tens of thousands of cycles, rather than the billions of cycles used in primary RTL verification.

A shadow-mode simulation using circuits from the failing unit still ran flawlessly. However, an engineer observed that a long wire crossing a large schematic was driven from both ends to reduce the RC delay. The signals X1 and X2 driving each end were intended to be identical (Figure 15.24). The engineer experi-

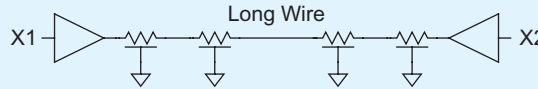


FIGURE 15.24 Long wire driven from both ends

mented with splitting the wire and checking that both drivers produced identical results, and on certain test cases they did not. This led to the wire experiencing contention and being driven to an indeterminate logic value. The invalid result propagated through other logic and hung the processor. Unfortunately, the circuit-understanding tool had incorrectly determined that the logic for the two ends was identical and had never detected the error. Even if the tool had been correct, the original test cases never would have exercised the patterns that caused the drivers to produce different results. A simple modification to the driver fixed the problem, but many units were already in the field. Fortunately, a software patch was developed to prevent the operations that caused the hang from ever being issued.

Hanging is a serious problem, but not as severe as unknowingly calculating the wrong answer. After the problem was corrected, engineers spent several more weeks proving to customers that the failure mode would hang the machine but could never result in an incorrect calculation.

To avoid repeating this problem in the future, engineers have turned to formal verification tools that prove that RTL and schematics are equivalent in their Boolean function. Such tools are not susceptible to incomplete test patterns. However, the tools are often expensive, proprietary, and difficult to use.

A product fails after the manufacturing process matures

A team designing a data communications product was comfortable with a particular microprocessor that was at the end of its production run. The team negotiated to order several thousand units of the discontinued microprocessor before production was shut down. The data communications product became successful and was shipped in large quantity. After it had been in the field for some time, major customers reported that the product would crash in large networks. These customers included large financial, government, and Internet service provider organizations who were adversely affected by the crashes. It took the data communications company weeks to isolate the problem to hanging of the microprocessor, and then a team of engineers at the microprocessor company began investigating the issue.

The microprocessor team investigated potential signal and power supply integrity issues. Although no signal integrity problems were apparent, a shmoo plot showed unusual sensitivity of minimum clock period to supply voltage. An engineer had recently read the application note for the power regulator on the system board and had learned that it had a propensity for oscillation if not properly bypassed. The system board lacked the bypass capacitors recommended in the application note, so the engineer wrote a memo to the product manager suggesting a change to the board. The memo was misinterpreted as a solution to the problem and customers were informed that a fix was on its way. Unfortunately, further testing showed that bypassing the regulator did not fix the crashes.

When the system crashed, it wrote its state to a core file. An engineer began reading a hexadecimal dump of the file and noticed a pattern that led to solving the crash. The pattern was associated with simultaneous access to many banks in an eight-way associative instruction cache. The cache had fuses associated with each bank, so banks containing bad blocks could be disabled during manufacturing test. During original product debug, the manufacturing process was relatively immature and most processors only had five operational cache banks. However, the processors manufactured at the end of the production run were built on a more mature process and often had all eight banks functional. Simultaneous access to all the banks tickled a signal integrity problem, resulting in power supply droop from excessive IR drops caused by poor contacts to the V_{DD} plane. The solution was a software change to disable three of the banks at system startup.

Better power supply analysis is performed to avoid repeating this problem.

A wasted spin

A microprocessor was taped out and came back nearly fully operational. Minor changes were made to the layout and documentation was developed; then a second revision (colloquially called a second *spin* of the chip) was taped out. The second revision came back completely non-functional, with a short between power and ground. Optical inspection while manufacturing the polysilicon layer showed that there was no field oxide on the chip.

Inspection of the masks showed that the active area mask specified active area (i.e., diffusion) for the entire chip rather than just where transistors belonged. The layout tool assigned each layer—such as active area or metal1—a unique number. However, although the layout for active area layer was correct, the mask did not appear to match the active layer.

Layout documentation had been annotated on an unused layer by drawing rectangles and text to indicate functional blocks. A larger rectangle defined the entire chip area. Careful tracing of the mask-generation software found that the “unused” layer had been used for active area many years ago and that the documentation rectangles were merged with the true active area to form a blob of active covering the entire chip.

Another microprocessor from a different vendor also failed when it was first built. Visual inspection of the die showed that the entire cache was missing. The cache had been removed from the design database to speed up final verification because it had already been checked separately. An engineer neglected to put it back in before tapeout.

Both of these wasted fabrication runs could have been avoided by using more rigorous verification methods at both the design and mask fabrication facilities. Validation of dataset size by the designer would have caught the missing geometries. Use of the industry standard mask database inspection tools would have caught the error after mask build. Although in the past, fabrication of a modest number of parts for testing was a small part of the design cost, with the escalation of mask and wafer fabrication costs, these mistakes can be a multimillion-dollar error. The extra time to market has a large opportunity cost as well.

At high voltage, a chip only operates at low frequency

While booting the operating system during silicon debug, a microprocessor operated as expected at low voltage. At high voltage, the part only functioned at low frequency. The high-voltage roof is an indication of a potential coupling problem in which the coupling is exacerbated by the fast edge rates associated with high-voltage operation. Test cases revealed that the problem resulted from incorrect operation of the register file when certain instructions executed. When the designers inspected the scan latches, they found that the correct 0 value was sent to the register file to write, but that an incorrect 1 was read. This indicated that either read or write operation was failing at high voltage. Trying one operation at high voltage and the other at low voltage proved the problem was in the write path.

A schematic of the register file write circuitry is shown in Figure 15.25. The register file uses predischarged write bitlines that are conditionally pulled high, depending on the data. The appropriate cell is written by turning on the corresponding write access transistor. The register cell is intentionally unstable so that the value on the bitline can overpower the cell and write the appropriate value. A weak keeper holds the metal2 bitline low when writing a 0. However, the register file is large and the keeper is at the opposite end from the data transistor. The resistance of the long, thin wire further reduces the effectiveness of the keeper against noise on the bitline.

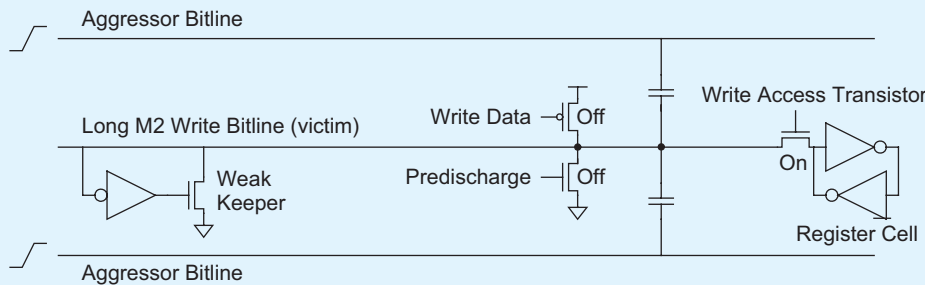


FIGURE 15.25 Register file write circuitry

When the neighboring bitlines switch high, they couple onto the victim line and tend to pull it high. The circuit fails if the aggressors introduce too much coupling noise. At high voltage, the aggressor drivers are stronger and cause a momentary glitch on the victim. At low frequency, the keeper is sufficient to restore the victim to a low level.

The coupling problem had been flagged during design by an automated noise-checking tool. However, the tool is conservative and the area of the register file would have increased significantly if the bitlines were spaced far enough apart to satisfy the tool. Therefore, the designer checked for excessive coupling with a SPICE simulation. The simulation apparently did not properly model the combination of circumstances that caused the failure. A second engineer cross-checked all circuits that waived the noise-checker warning, but also did not discover the excessive coupling. The problem was solved by placing a second keeper near the write data transistor to fight against the coupling.

Another funny shmoo

During silicon debug, a microprocessor cache only functioned correctly over the peculiar range of voltages and frequencies shown in the shmoo¹ in Figure 15.26. Test code exercising the

¹A shmoo of this type is sometimes called a *flying saucer*.

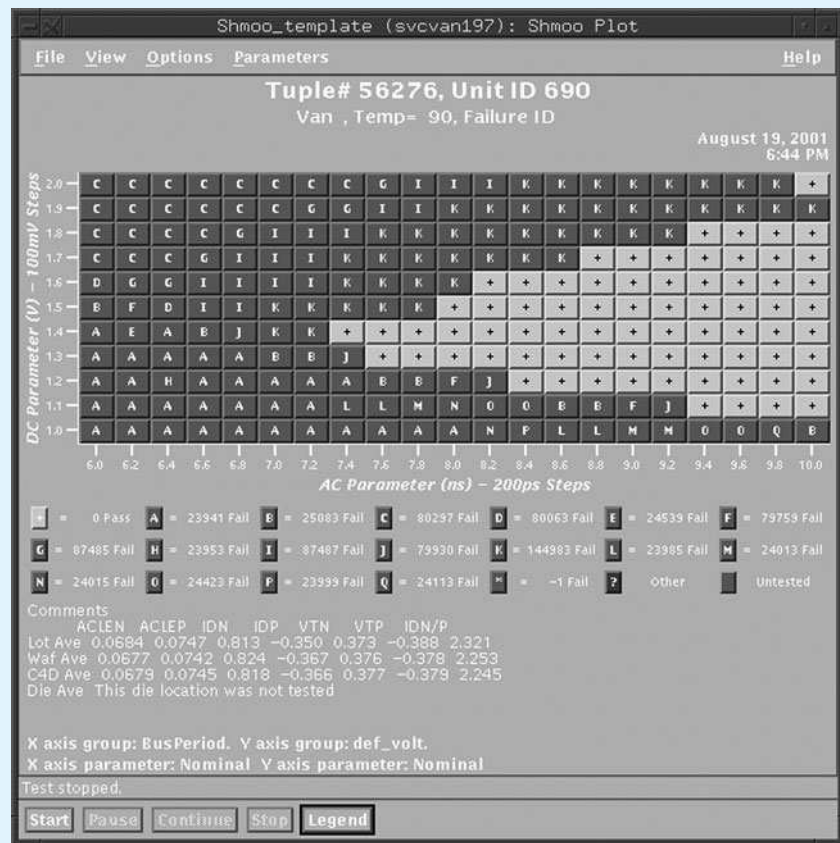


FIGURE 15.26 Flying saucer shmoo

cache revealed that failures were caused by bad data being read from the cache. Scan isolated the problem to a dynamic multiplexer choosing one of the global bitlines, as shown in Figure 15.27.

The multiplexer inputs were the NORs of dynamic metal3 global bitlines and corresponding select signals. The metal4 select lines were early and did not need to be dynamic, but were implemented as dynamic nodes anyway. All of the transistors in the dynamic multiplexer were supposed to remain OFF in this particular test case, leaving the multiplexer output high.

One input of the multiplexer had a low value on the global bitline, but was not selected, as shown. Therefore, the transistor should have been OFF. Nevertheless, the output of the multiplexer incorrectly discharged. One neighbor of the select line was ground; the other fell low. Coupling from a single neighbor is generally not enough to cause noise failure. However, many global bitlines ran over the top of the select line and also fell low. Laser voltage probing showed

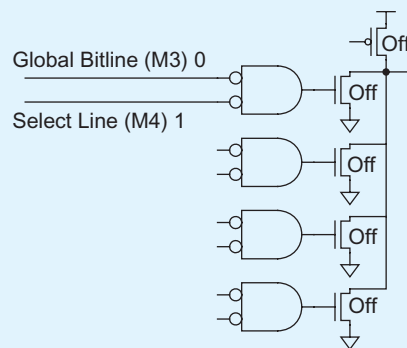


FIGURE 15.27 Dynamic bitline multiplexer

that the select line was incorrectly pulled low, apparently from coupling caused by these falling bitlines as well as the neighbor line. The odd shape of the shmoo happened because the failures only occurred when the neighbor and overhead lines both fell at about the same time; otherwise, the keeper on the select line was strong enough to recover from one noise event before the other arrived. Because the bitline and control paths were different, the noise events only happened simultaneously for certain voltages.

Noise analysis tools usually check only neighbors, and the single switching neighbor was not sufficient to trigger an error. In this circumstance, so many global bitlines ran over the top of the select wire that their coupling could not be neglected. The problem was fixed by converting the control line into a static signal more resistant to coupling noise. A better noise analyzer could have considered coupling from neighbors above and below, especially on dynamic nets. However, it is difficult to extract information about such orthogonal neighbors because they are often drawn at different levels of the layout hierarchy. Moreover, assuming all neighbors switch in the worst possible direction is usually pessimistic for long wires. Nevertheless, such a data-dependent failure mechanism is a source of nightmares for designers.

Incorrect operation at low temperature

A floating-point coprocessor was tested by running the LINPACK benchmark. The benchmark performs a series of floating-point operations and generates a checksum to verify the result. The chip would occasionally produce the wrong checksum. One of the engineers heated the coprocessor by removing the heat sink and found that the coprocessor became reliable at higher temperature.

This suggested that the problem might be caused by coupling, which is generally more serious at lower temperature where the edge rates are faster. The error was tracked to a long on-chip bus with many wires laid out on a tight pitch. Although the wires were subject to coupling noise, they were not on a critical path and should have had plenty of time to settle to the correct value. Unfortunately, they drove the diffusion input of a latch. When crosstalk drove an input below $-V_t$, it would turn on the pass transistor and incorrectly discharge the latch (see Section 9.3.9).

The floating-point unit bug was holding up lucrative product shipments. While a corrected coprocessor was being fabricated, the old unit was shipped in products with a bolt-on thermostat/heater unit used to guarantee a minimum operating temperature.

An obvious lesson of this experience is to avoid driving diffusion inputs with potentially noisy signals. More fundamentally, however, this bug demonstrated a marginal design of the cell library that should have been caught in the library review. Moreover, humans are inherently prone to errors. Electrical rules like no noisy diffusion inputs aren't worth the paper they are printed on unless computer code exists to enforce them.

Slower than expected performance

An application-specific integrated circuit (ASIC) was fabricated on a gate array by a third-party gate array manufacturer. Although static timing analysis predicted that the chip would function fast enough, the manufacturer found that most of the chips would not operate at the desired frequency and instead had to be derated by about 20%.

The designer examined a die plot, looking for the source of the unexpectedly slow performance. The plot showed that the horizontal power and ground lines were only strapped along the edges of the chip, as shown in Figure 15.28(a). Some rows of gates consumed large amounts of power, causing large IR drops along their power lines. Measurements showed that the power supply sometimes drooped below 2 V, despite the nominal 3.3 V power supply. When the wide vertical power supply straps were added, as shown in Figure 15.28(b), most chips met target speed.

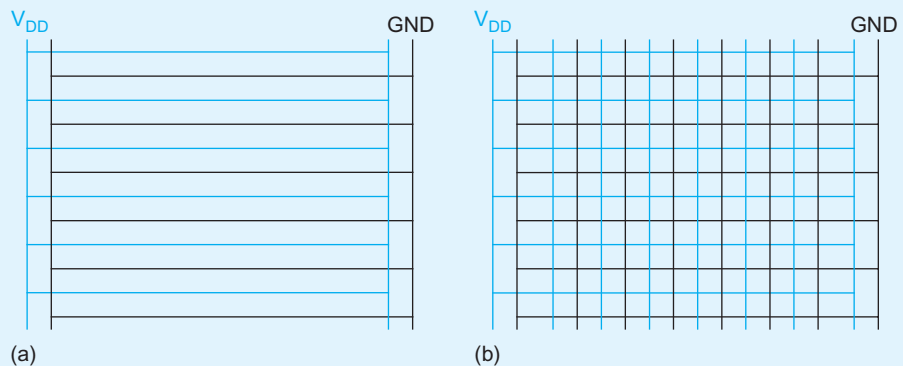


FIGURE 15.28 Power supply network

Modern chips require low-resistance on-chip power distribution networks and often use power and ground pads distributed across the die rather than just at the periphery to reduce the distance and resistance between the pads and the gates. Power integrity analysis should be performed to verify that the static or dynamic voltage droops remain within their budget everywhere on the chip.

Class chip failures

One of the authors has supervised a number of class project chips. The following are some of the reasons that chips have come back partially or completely nonfunctional:

- *Insufficient simulation*

A ring oscillator was placed on the chip as a test structure to verify that the hardware was at least partially functional even if the rest of the chip might not work. It didn't oscillate. It had not been simulated because it was "too simple." Inspection during debug found that the oscillator had an even number of inverters!

Another chip was designed with a new CAD tool that had a buggy simulator. Most of the chip operated correctly, but the chip as a whole would not simulate. The problem was attributed to a bug in the simulator and was taped out anyway. The chip came back nonfunctional.

- *Incomplete top-level verification*

One year, a pad frame was used that was incompatible with the normal verification flow. The chip cores were verified, placed in the pad frame, and then routed to the pads. DRC and simulation were not performed on the connections to the pads, so students carefully scrutinized their routing by hand. Upon testing, three of the four different designs were found to have errors in the routing to the pads. No errors were found in the cores that had been verified. "If you don't test it, it won't work! (guaranteed)."

- A neural network chip seemed to have a defective scan chain because the scan data out line never budged from 0 as configuration data was scanned into the chip. Testing found that the chip was correctly configured except in the last bit of the scan chain. Inspection of the layout revealed that the scan data out line (which came from the last bit of the scan chain) had been shorted to ground while being routed to the pads.

- A carry-lookahead adder produced incorrect results on certain input patterns. The least significant bits were always correct. Inspection of the layout revealed that the A[4] input was routed from the pad most of the way to the core but part of the wire was missing, probably because the designer accidentally hit UNDO after finishing the route.
- A GPS searcher chip had an inverter connected to a pair of pins to verify that the chip showed basic functionality. The output was stuck low. Inspection of the layout revealed that the input was attached to an output pad and the output to an input pad. The GPS searcher itself was fully operational.

While some of these may represent class situations, the same type of reasons for partial failure also plague industry chips. In particular, when time scales are stressed, the boundary conditions are often overlooked, which leads to problems when the chips are fabricated. Once a good verification methodology is put in place that includes a known-good pad frame, top-level DRC, and full-chip simulation, students have had a 100% success rate on class chips.

Summary

This chapter has summarized the important issues in CMOS chip testing and has provided some methods for incorporating test considerations into chips from the start of the design. Scan is now an indispensable technique to observe and control registers because probing signals directly has become extremely difficult. The importance of writing adequate tests for both the functional verification and manufacturing verification cannot be understated. It is probably the single most important activity in any CMOS chip design cycle and usually takes the longest time no matter what design methodology is used. If one message is left in your mind after reading this chapter, it should be that you are absolutely rigorous about the testing activity surrounding a chip project and it should rank first among any design trade-offs.

Exercises

- 15.1 A circuit does not operate at the desired frequency. Cooling the circuit with freeze spray fixes the problem. A shmoo shows the circuit operates correctly at higher than nominal V_{DD} . What is the general nature of the likely problem and why?
- 15.2 You have to test a large die (1 cm × 1 cm) that is housed in a package that costs \$5. Would you do wafer testing? Why?
- 15.3 A verification script detects a single discrepancy between the golden model and your design out of 400,000 vectors. Would you proceed to fabrication? Explain your decision.
- 15.4 Explain what is meant by a Stuck-at-1 fault and a Stuck-at-0 fault.
- 15.5 How are sequential faults caused in CMOS? Give an example.
- 15.6 Explain the different kinds of physical faults that can occur on a CMOS chip and relate them to typical circuit failures.

- 15.7 Explain the terms controllability, observability, and fault coverage.
- 15.8 Why is it important to have a high fault coverage for a set of test vectors?
- 15.9 Explain how serial-scan testing is implemented.
- 15.10 Explain the principles of Built-In Self-Test (BIST). What are the advantages and disadvantages of BIST?
- 15.11 You have to design an extremely fast divide by eight frequency divider that taxes the capabilities of the process you are using. What test strategy would you employ to test the divider? Explain the reasons for your choice.
- 15.12 Design a register that minimizes transistor count, but allows parallel scan to be implemented, as outlined in Figure 15.17.
- 15.13 Explain how a Pseudo-Random Sequence Generator (PRSG) can be used to test a 16-bit datapath. How would the outputs be collected and checked?
- 15.14 Design a block diagram of a test generator for a $4K \times 32$ static RAM.
- 15.15 Research the origin of the term “shmoo.”