DFT: Design for Testability

When you buy a new piece of equipment, you expect it to work properly "right out of the box." However, even with modern automated equipment, it's impossible for an equipment manufacturer to guarantee that every unit produced will be perfect. Some units don't work because they contain individual components that are faulty, or because they were assembled incorrectly or sloppily, or because they were damaged in handling. Most manufacturers prefer to discover these problems in the factory, rather than get a bad reputation when their customers receive a faulty product. This is the purpose of *testing*.

testing

The most basic test, a *go/no-go test*, yields just one bit of information—is the system 100% functional or not? If the answer is yes, the system can be shipped. If not, the next action depends on the size of the system. The appropriate response for a digital watch would be to throw it out. If many watches are turning up faulty, it's more efficient to repair what must be a consistent problem in the assembly process or an individual component than to try to salvage individual units.

go/no-go test

On the other hand, you wouldn't want to toss out a $10,000 computer that fails to boot. Instead, a more detailed *diagnostic test* may be run to locate the particular subsystem that is failing. Depending on cost, the failed subsystem may be either repaired or replaced. A typical digital subsystem is a PCB whose assembled cost with components ranges from $50 to $1,000. The repair/replace decision is an economic trade-off between the cost of the assembled PCB and the estimated time to locate and repair the failure.

diagnostic test

This is where *design for testability (DFT)* comes in. "DFT" is a general term applied to design methods that lead to more thorough and less costly testing. Many benefits ensue from designing a system or subsystem so that failures are easy to detect and locate:

design for testability (DFT)

- The outcome of a go/no-go test is more believable. If fewer systems with hidden faults are shipped, fewer customers get upset, which yields obvious economic as well as psychological benefits.
- Diagnostic tests run faster and produce more accurate results. This reduces the cost of salvaging a subsystem that fails the go/no-go test, making it possible to manufacture more systems at lower cost.
- Both go/no-go and diagnostic tests require less test-engineering time to develop.
- Although the savings in test-engineering time may be offset by added design-engineering effort to include DFT, any increase in overall product development cost usually can be offset by decreased manufacturing cost.

DFT.1 Testing

Digital circuits are tested by applying *test vectors* which consist of input combinations and expected output combinations. A circuit "passes" if its outputs match what's expected. In the worst case, an n-input combinational circuit requires 2^n test vectors. However, if we know something about the circuit's physical realization and make some assumptions about the type of failures that can occur, the number of vectors required to test the circuit fully can be greatly reduced. The most common assumption is that failures are *single stuck-at faults*, that is, that they can be modeled as a single input or output signal stuck at logic 0 or logic 1. Under this assumption, an 8-input NAND gate, which might otherwise require 256 test vectors, can be fully tested with just nine—11111111, 01111111, 10111111, …, 11111110.

test vector

single stuck-at fault

Under the single-fault assumption, it's easy to come up with test vectors for individual logic elements. However, the problem in practice is *applying* the test vectors to logic elements that are buried deep in a circuit, and *seeing* the results. For example, suppose that a circuit has a dozen combinational and sequential logic elements between its primary inputs and the inputs of an 8-input NAND gate that we want to test. It's not at all obvious what primary-input vector, or sequence of primary-input vectors, must be applied to generate the test vector 11111111 at the NAND-gate inputs. Furthermore, it's not obvious what else might be required to propagate the NAND gate's output to a primary output of the circuit.

Sophisticated *test-generation programs* deal with this complexity and try to create a *complete test set* for a circuit, that is, a sequence of test patterns that fully tests each logic element in the circuit. However, the computation required can be huge, and it's quite often just not possible to generate a complete test set.

test-generation program

complete test set

DFT methods attempt to simplify test-pattern generation by enhancing the "controllability" and "observability" of logic elements in a circuit. In a circuit with good *controllability*, it's easy to produce any desired values on the internal signals of the circuit by applying an appropriate test-vector input combination to the primary inputs. Similarly, good *observability* means that any internal signal can be easily propagated to a primary output for comparison with an expected value by the application of an appropriate primary-input combination. The most common method of improving controllability and observability is to add *test points,* additional primary inputs and outputs that are used during testing.

controllability

observability

test points

DFT.2 Bed-of-Nails and In-Circuit Testing

In a digital circuit that is fabricated on a single PCB, the "ultimate" in observability is obtained by using every pin of every IC as a test point. This is achieved by building a special *test fixture* that matches the layout of the PCB and contains a spring-loaded pin (*nail*) at each IC-pin position. The PCB is placed on this *bed of nails,* and the nails are connected to an *automatic tester* that can monitor each pin as required by a test program.

test fixture

nail

bed of nails

automatic tester

Going one step further, *in-circuit testing* also achieves the "ultimate" in controllability. This method not only monitors the signals on the bed of nails for observability, but also connects each nail to a very low impedance driving circuit in the tester. In this way, the tester can override (or *overdrive*) whatever circuit on the PCB normally drives each signal, and directly generate any desired test vector on the internal signals of the PCB. Overdriving an opposing gate output causes excessive current flow in both the tester and the opposing gate, but the tester gets away with it for short periods (milliseconds).

in-circuit testing

overdrive

To test an 8-input NAND gate, an in-circuit tester needs only to provide the nine test vectors mentioned previously, and can ignore whatever values the rest of the circuit is trying to drive onto the eight input pins. And the NAND-gate output can be observed directly on the output pin, of course. With in-circuit testing, each logic element can be tested in isolation from the others.

Although in-circuit testing greatly enhances the controllability and observability of a PCB-based circuit, logic designers must still follow a few DFT guidelines for the approach to be effective. Some of these are listed below.

- *Initialization.* It must be possible to initialize all sequential circuit elements to a known state.

Since the preset and clear input pins of registers and flip-flops are available to an in-circuit tester, you would think that this is no problem. However, Figure DFT-1(a) shows a classic example of a circuit (a Gray-code counter) that cannot be initialized, since the flip-flops go to an unpredictable state when PR and CLR are asserted simultaneously. The correct way to handle the preset and clear inputs is shown in (b).

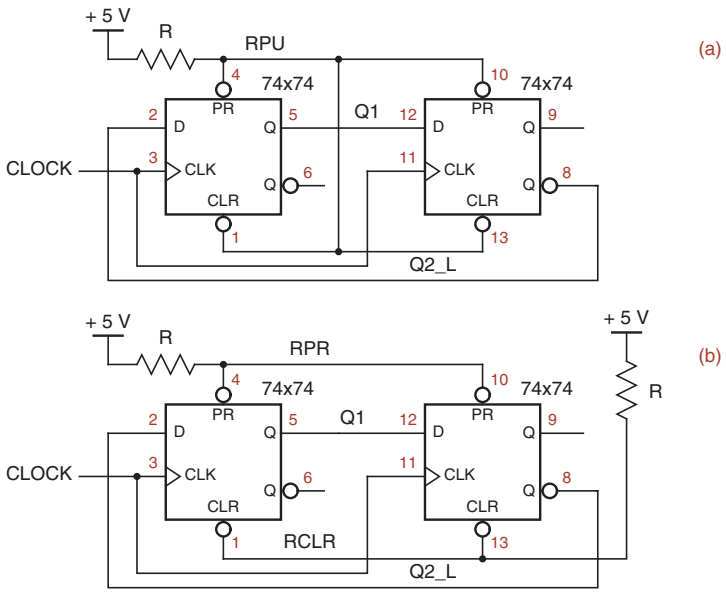

Figure DFT-1
Flip-flops with pull-up resistors for unused inputs: (a) untestable; (b) testable.

HITTING THE NAIL ON THE PIN

The rise of high-density surface-mount packaging has made bed-of-nails testing considerably more difficult than it was with through-hole components like DIPs. Since components may be mounted on both sides of the PCB, a special test fixture called a clam shell may be needed to connect nails to both sides of the PCB.

Furthermore, the pins of many surface-mount devices are so small and their spacing is so tight (25 mils or less), that it may be impossible to reliably land a test pin on them. In such cases, the PCB designer may have to explicitly provide test pads, extra copper patches that are large enough for a test pin to contact (e.g., 50 mils in diameter). A separate test pad must be provided for each signal that does not connect to a larger (e.g., 62-mil through-hole) component pad somewhere on the PCB.

- *Clock generation.* The tester must be able to provide its own clock signal *without* overdriving the on-board clock signals.

Testers usually must override an on-board clock, for several reasons: The speed at which they can apply test vectors is limited; they must allow extra time for overdriven signals to settle; and sometimes they must stop the clock. However, overdriving the clock is a no-no. An overdriven signal may "ring" and make several transitions between LOW and HIGH before finally settling to the level that the tester wants. On a clock signal, such transitions can create unwanted state changes.

Figure DFT-2 shows a recommended clock driver circuit. To inject its own clock, the tester pulls CLKEN LOW and inserts its clock on TESTCLK_L. Since the tester is not overdriving any gate outputs, the resulting CLOCK signal is clean. In general, any normally glitch-free signal that is used as a clock input or other asynchronous input must not be overdriven by the tester, and would have to be treated in a way similar to Figure DFT-2. This is another reason why synchronous design with a single clock is so desirable.

- *Grounded inputs.* In general, ground should not be used as a constant-0 logic source.

The in-circuit tester can overdrive *most* signals, but it can't overdrive ground. Therefore, signals that require a constant-0 input during normal operation

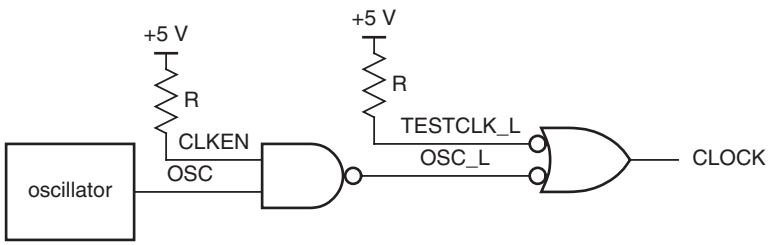

Figure DFT-2
Clock driver circuit that allows a tester to cleanly override the system clock.

should be tied to ground through a resistor, which allows the tester to set them to 1 as required during testing. For example, consider what would happen if we used a GAL16V8 PLD to generate a customized set of clock waveforms from a master clock, as we did in the ABEL program of Table 8-31 on page 744. If not for testing considerations, we could connect the 16V8's pin 11, its active-low global output-enable pin, directly to ground. However, it should be connected through a resistor, so that the tester can float the PLD outputs and drive the clock signals P1_L–P6_L itself. Although the tester could theoretically overdrive these signals, it shouldn't if they are used as clocks.

- *Bus drivers.* In general, it should be possible to disable the drivers for wide buses so that the tester can drive the bus without having to overdrive all the signals on the bus.

That is, it should be possible to output-disable all of the three-state drivers on a bus, so that the tester drives a "floating" bus. This reduces electrical stress both on the tester and on multibit drivers (e.g., 74x244) that might otherwise be overheated and damaged by having all of their outputs overdriven simultaneously.

DFT.3 Scan Methods

In-circuit testing works fine, up to a point. It doesn't do much good for custom VLSI chips and ASICs, because the internal signals simply aren't accessible. Even in board-level circuits, high-density packaging technologies such as surface mounting greatly increase the difficulty of providing a test point for every signal on a PCB. As a result, an increasing number of designs are using "scan methods" to provide controllability and observability.

scan method
scan-path method

A *scan method* attempts to control and observe the internal signals of a circuit using only a small number of test points. A *scan-path method* considers any digital circuit to be a collection of flip-flops or other storage elements interconnected by combinational logic, and is concerned with controlling and observing the state of the storage elements. It does this by providing two operating modes: a normal mode, and a *scan mode* in which all of the storage elements are reorganized into a giant shift register. In scan mode, the state of the circuit's *n* storage elements can be read out by *n* shifts (*observability*), and a new state can be loaded at the same time (*controllability*).

scan mode

Figure DFT-3 on the next page shows a circuit designed using a scan-path method. Each storage element in this circuit is a scan flip-flop (Section 7.2.7) that can be loaded from one of two sources. The test enable (TE) input selects the source—normal data (D) or test data (T). The T inputs are daisy-chained to create the scan path shown in color. By asserting ENSCAN for 11 clock ticks, a tester can read out the current state of the flip-flops and load a new state. The test engineer is left with the job of deriving test sets for the individual combinational logic blocks, which can be fully controlled and observed using the scan path and the primary inputs and outputs.

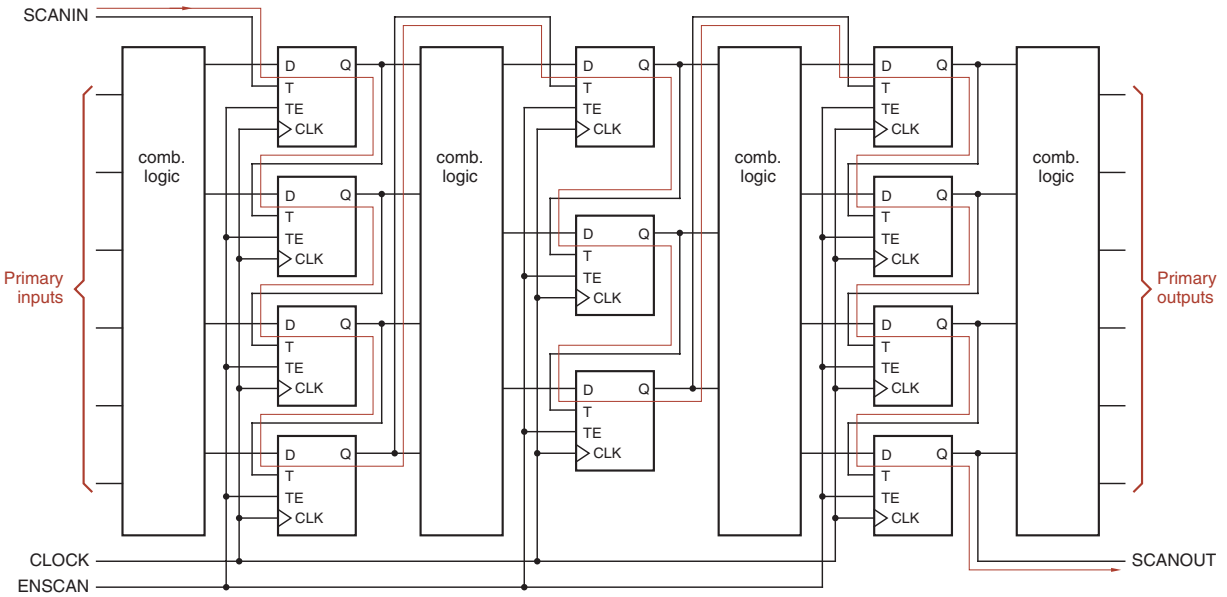

Figure DFT-3 Circuit containing a scan path, shown in color.

Scan-path design is used most often in custom VLSI and ASIC design, because of the impossibility of providing a large number of conventional test points. However, the two-port flip-flops used in scan-path design do increase chip area. For example, in LSI Logic Corp.'s LCA500K series of CMOS gate arrays, an FD1QP D flip-flop macrocell uses seven "gate cells," while an FD1SQP D scan flip-flop macrocell uses nine gate cells, almost a 30% increase in silicon area. However, the overall increase in chip area is much less, since flip-flops are only a fraction of the chip, and large "regular" memory structures (e.g., RAM) may be tested by other means. In any case, the improvement in testability may actually *reduce* the cost of the packaged chip when the cost of testing is considered. For large ASIC designs with rich, complicated control structures, scan-path design should be considered a requirement.

References

We briefly discussed device testing in the context of ABEL test vectors. There is a large, well-established body of literature on digital device testing, and a good starting point for study is McCluskey's 1986 book. Generating a set of test vectors that completely tests a large circuit such as a PLD is a task best left to a program. At least one company's entire business is focused on programs that automatically create test vectors for PLD testing (ACUGEN Software, Inc., Nashua, NH 03063, `www.acugen.com`).