4. Testing

1

- Defects, complexity, fault models, coverage.
- Test pattern generation.
- Scan path and built-in self-test. Boundary scan.
- Other issues: delay-fault testing, functional testing.

References

- J M Rabaey et al, “Digital Integrated Circuits”, 2nd edition Prentice Hall 2003.
- E J McCluskey, “Logic Design Principles”, Prentice Hall 1986.
- S L Hurst, “VLSI Testing” IEE 1998.
- B R Wilkins, Testing Digital Circuits, VNR 1986.

2

If you can get hold of it, Rabaey is probably the most appropriate to this course. The book by Smith recommended for B3 is also useful.

Although out of print, McCluskey is also good an may be more widely available in libraries.

Hurst is good but has much more detail than is needed for the course.

Wilkins is a simple text which may be helpful if you can’t find the others.

There is a also a brief summary in Wakerly Section 11.2.

Why is Testing Needed?

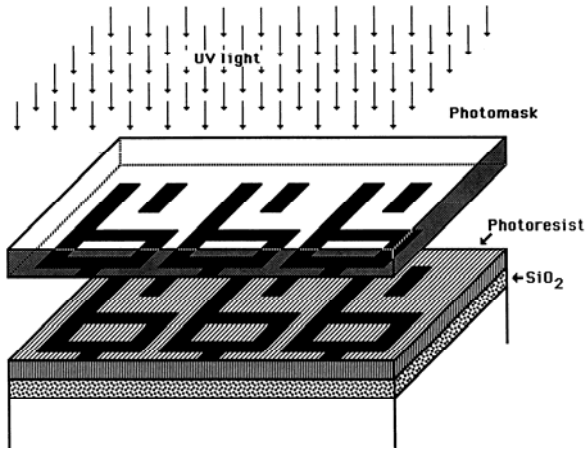
3

Designs don’t always do what you want.

ICs & PCBs do not always work!

Things fail.

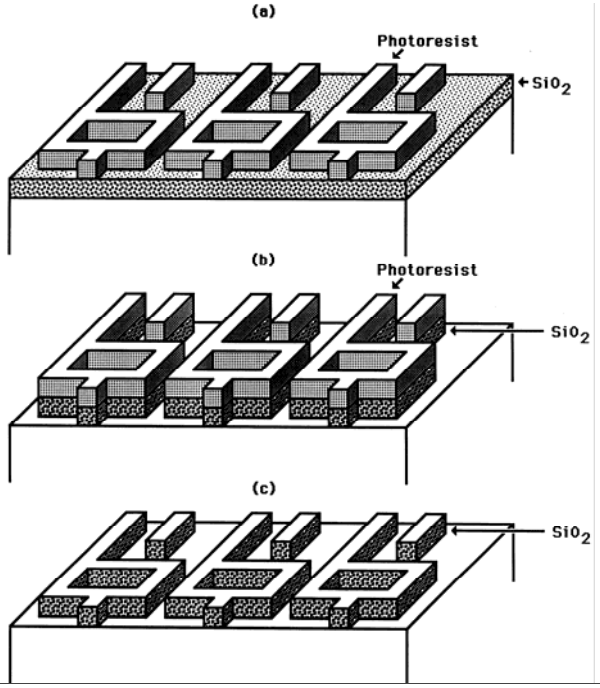
Photolithography - 1



4

Your CAD tools produce nice neat shapes

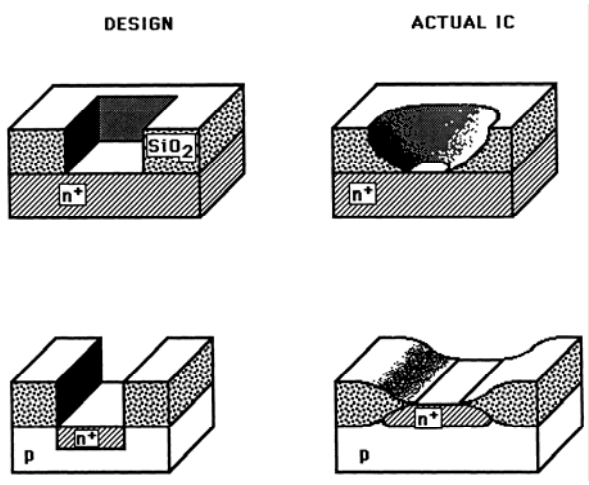
Photolithography - 2



5

… and you may be forgiven for thinking that ICs look like this.

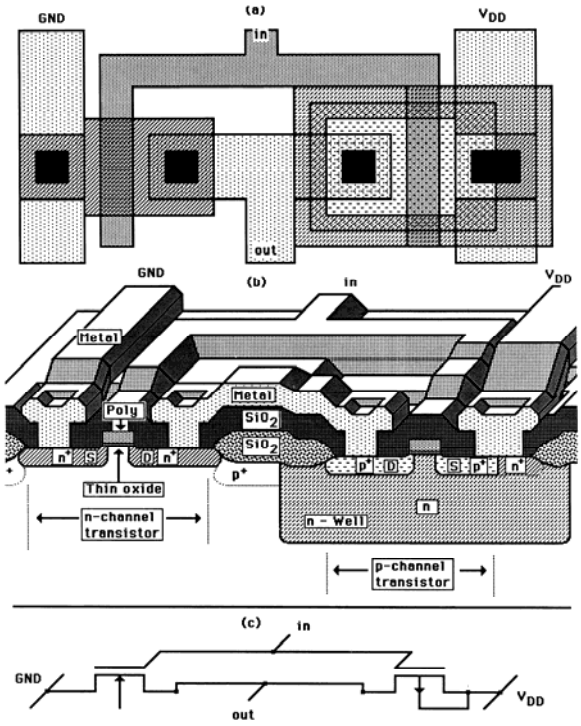
Physics & Chemistry!



6

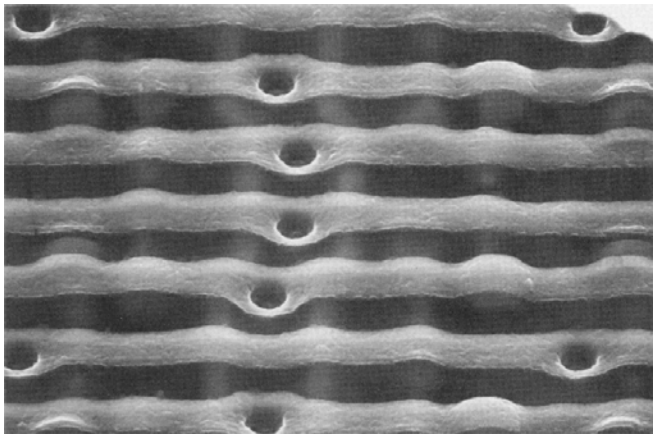
But actually it’s a bit more complicated!

Processing is 3-D



This CMOS inverter shows how tortuous things get in 3-D.

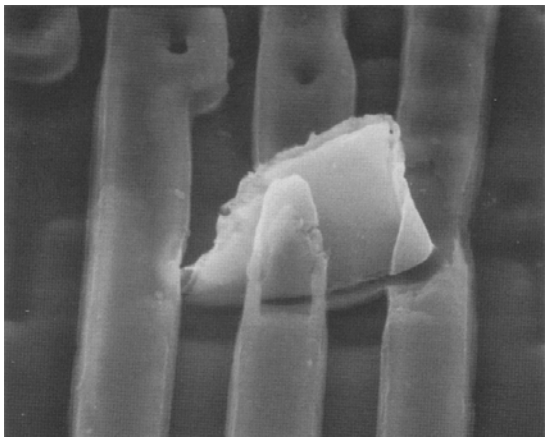
Real Life



8

In real life even simple things get very messy. Look at these parallel conductors under a microscope.

Oops!



9

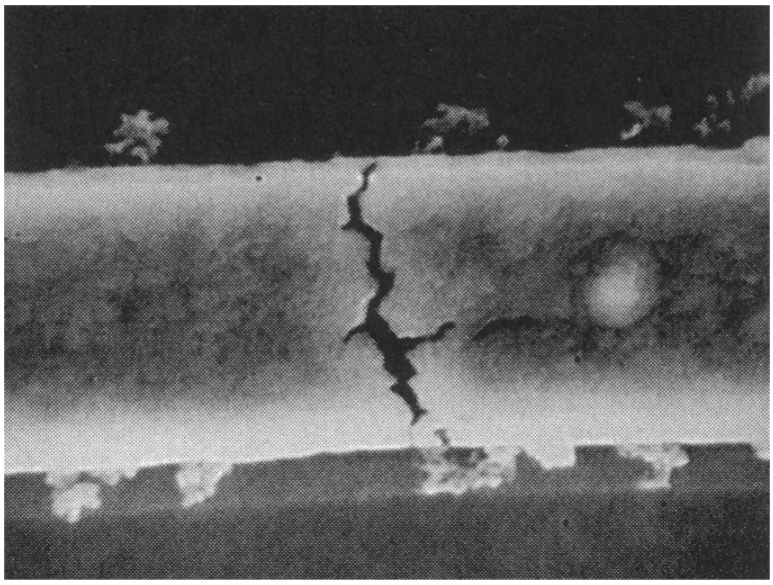
…and worse!

In the production of aggressive VLSI designs, a *yield* (i.e. the proportion of manufactured ICs which actually work) of 60% is considered to be good. Designs have certainly been put into production with much lower yields (maybe only 20%!) and when the yield reaches 80% or higher, it's probably time to make something more complicated.

Here we see a “bridging fault” due to some contamination.

Similarly it is unrealistic to expect that all PCBs or electronic systems will work first time (though generally the yields are substantially better than those of ICs!)

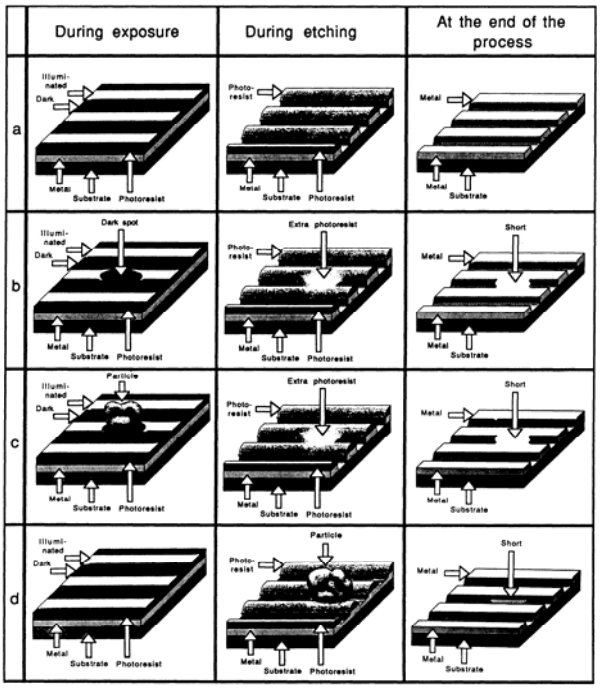
Oh Dear!



10

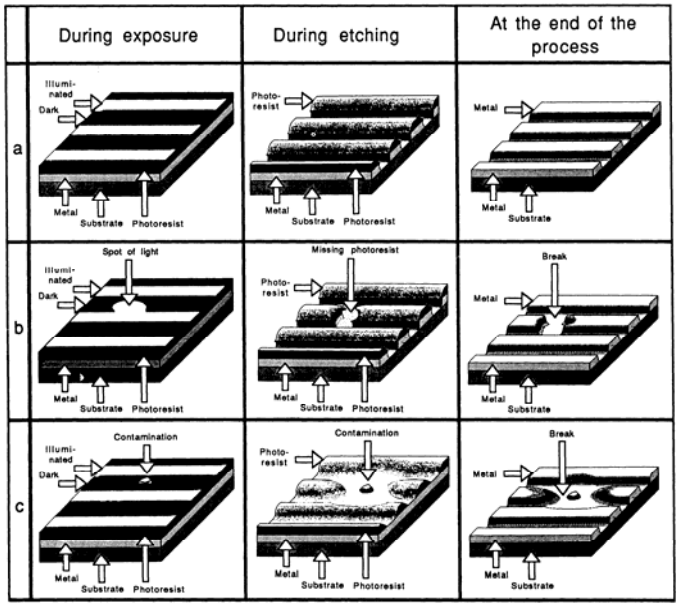
.. and here is an “open circuit” fault.

Some Bridging Fault Mechanisms



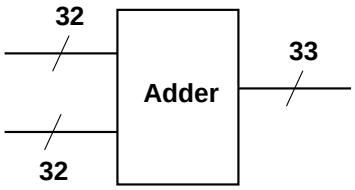
Here are some things that can cause bridges between metal lines metal lines.

Some Open Circuit Fault Mechanisms



… and open circuits.

Why is Testing a Problem?



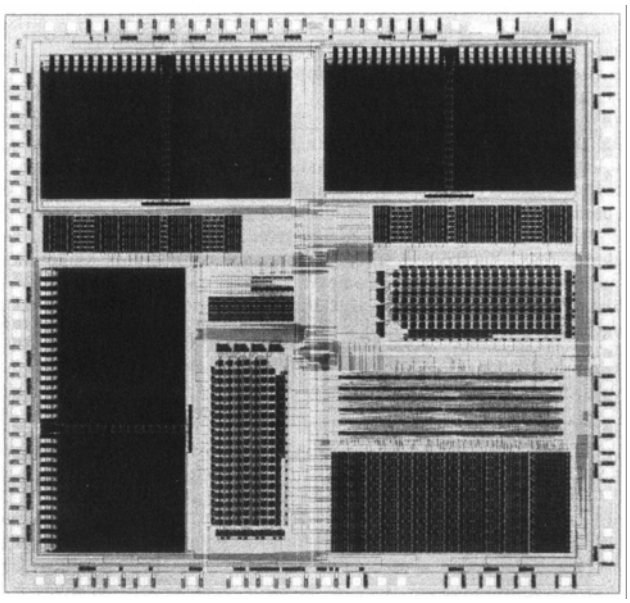
13

Consider a combinational circuit with *n* inputs.

As you found out last year in B3, there are 2^n possible input patterns that you could apply to it. Therefore *exhaustive* testing of an adder with two 32-bit inputs would take 2^{64} tests. Even at 100MHz (typical of the best testers available – with > $1M price tag) this is almost 6000 years!

Worse than that(!) for a sequential circuit the situation with *m* binary state variables (flip-flops) the number of possible tests is 2^{n+m} *and* you also have the problems of how to set the flip-flops and how to observe their values at the output!

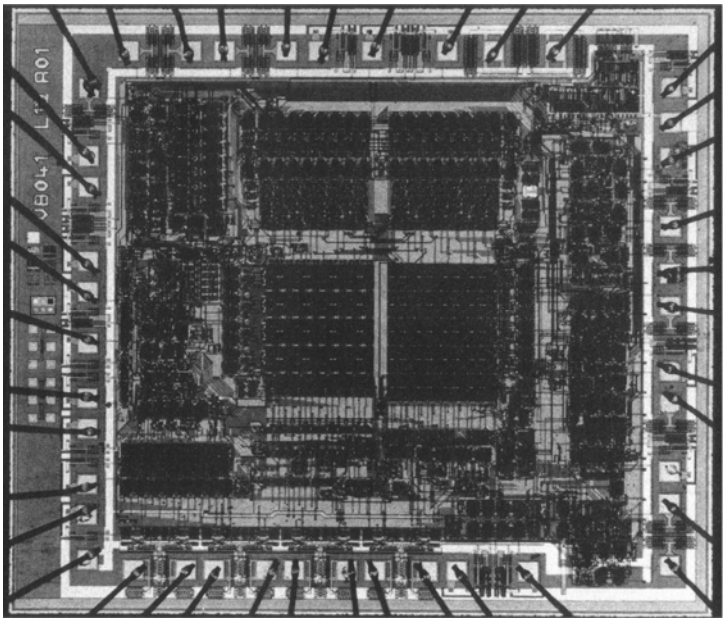
What's wrong with this chip?



14

Suppose just one transistor is defective, how can we find out?

Or this one?



15

Clearly we have to make some assumptions about the sorts of things that are really likely to go wrong and test for these. Although we can in this way dramatically reduce the test time, this reduction comes at the cost either of some pretty complicated and long-winded test pattern generation (costly in design time), or else on some pretty strict design constraints (costly in circuit area).

In mass production, the test application time can also be very costly; for example an IC tester capable of applying a million test patterns of testing at 100MHz to 256 pin chips is likely to cost more than a million pounds. Testing of an IC must therefore usually be kept to at most a fraction of a second.

Cheaper testers will probably be much slower and be able to apply fewer test patterns (maybe as little as 1MHz and 2,000 patterns).

Definitions - 1

- Failure mechanism
- Fault
  - permanent (or hard)
  - transient (or soft).
- Error
- Failure
- Fault model
- Fault detection
- Fault diagnosis

16

The *failure mechanism* is the underlying physical problem that occurs.

The *fault* is the change that occurs to the circuit function; these can further be *permanent* (or *hard*) or they may be *transient* (or *soft*).

An *error* is an incorrect value produced by the faulty circuit.

Failure is the inability of our system to do what it is supposed to do.

A *fault model* is an assumption about the faults that are likely to occur.

Fault detection is the observation of some fault (through the presence of an error).

Fault diagnosis is the deduction of which particular fault has occurred, or at least which particular component is faulty.

Definitions - 2

- Fault coverage (fault grading)
- Testability
- Controllability
- Observability
- Test vector
- Test pattern
- Test (Sequence)

17

The *fault coverage* is the proportion of the possible faults that we are able to *detect; fault grading* is the exercise of assessing the fault coverage.

Testability is the ability to detect all the possible faults.

Controllability is the ability to excite the fault and produce an error .

Observability is the ability to *propagate* the error to an output.

A *test vector* is a set of input values applied to a circuit.

A *test pattern* is a set of input values and the expected output values.

A *test* (a *test sequence*) is a set of test patterns (in a specified order).

When do we test?

- Debug
- Production
- Field

18

Write your own notes.

What do we test for

- Diagnosis (improve/repair)
- Pass/fail (replace/discard)

19

Keep writing.

Test Costs

- Test generation
- Test evaluation
- Test application
- 1/yield
- Number of test stages:

20

Don’t stop …

Typical IC Tests

- Wafer probe test
- Packaged IC test

- Parametric test
- Voltage tests
- IDDQ test
- Delay tests

21

…now that you have the hang of it.

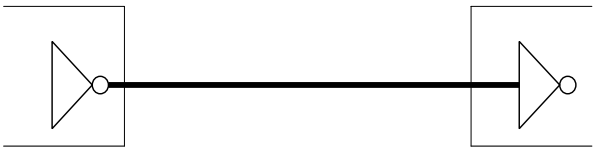
Typical PCB Tests

- Component
- Unpopulated board
- Populated board
- In-circuit test

22

Design for test: “ad hoc” techniques - 1

- Improve Observability



23

Add features to improve the "observability”:

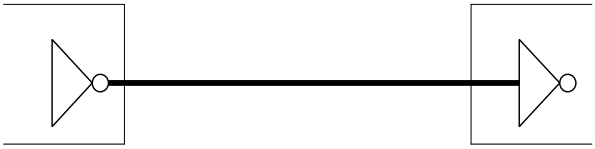
PCB e.g. bring more signals to the sockets, add test points with terminal post, test socket or flying lead.

PCB/IC e.g. multiplex key signals onto the outputs, add a serial scan path (see later).

Key observation points - "buried" logic, stored states, fan-in/fan-out nodes, feedback paths, monostables, redundant logic (perhaps put in to solve for race hazards, or to improve performance as in the use of complementary transistor pass gate in place of a single pass transistor, or because standard parts or cell library components have been used) etc.

“Ad hoc” techniques - 2

- Improve Controllability



24

Add features to improve the "controllability".
PCB e.g. inject test signal via test socket/flying lead.
insertion logic.
pull up resistors.
PCB/IC e.g. de-multiplex, scan path.

Key control points - clock, preset/clear, chip enables, tri-state controls, mode controls, read/write controls, busses, monostables, initialisation of complex parts, feedback paths, fan-in/fan-out nodes, etc.

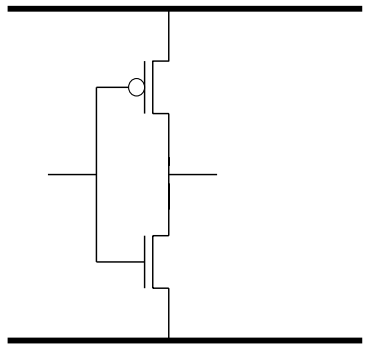
“Ad hoc” techniques - 3

- Avoid redundant logic
- Separate analogue/digital parts
- Partition circuit into manageable chunks
- Break feedback paths/long counter chains
- Avoid asynchronous logic (fundamental mode circuits and *especially* monostables)
- Avoid adjustable components.

25

Fault Models

- The single stuck-at model



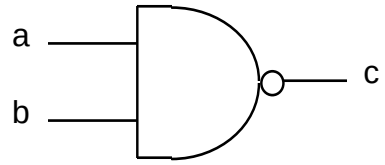
26

In order to make the test pattern generation problem tractable, we have to adopt a sensible “fault model”. This needs to be abstract enough that the test problem is simplified and realistic enough that the tests are effective.

If we model the 32-bit adder example above at the word level, we can deduce quite a lot by applying inputs of 23 *plus* 51 and seeing if the output is 76. For example this will tell us that the chip is actually connected, that the power supply is effective and that many of the pins are functioning. It will however by no means guarantee that all the connections are made or that all the transistors are working properly. If on the other hand we examine every transistor, resistor and their connections, we will we take a very long time to generate a test.

By far the most widely accepted model is the *single stuck-at* (SSA) fault model. In this we take a logic gate description of the circuit. We look at every single node of the circuit in turn we see what would happen if it were either stuck-at-0 or that it might be stuck-at-1. It turns out that a ripple carry adder circuit of *any width* can be 100% tested for single stuck-at faults with at most 8 test vectors. Of course it may often be that there is actually more than one fault in the circuit, but experience has shown that a test set which has a high coverage of single stuck-at faults will almost always have a high coverage of multiple faults. Problems can occasionally occur with *fault masking* but generally the more faults there are, the more likely it is that you will see that something is wrong.

Automatic Test Pattern Generation (ATPG)



a	b	c
1	1	0
0	x	1
x	0	1

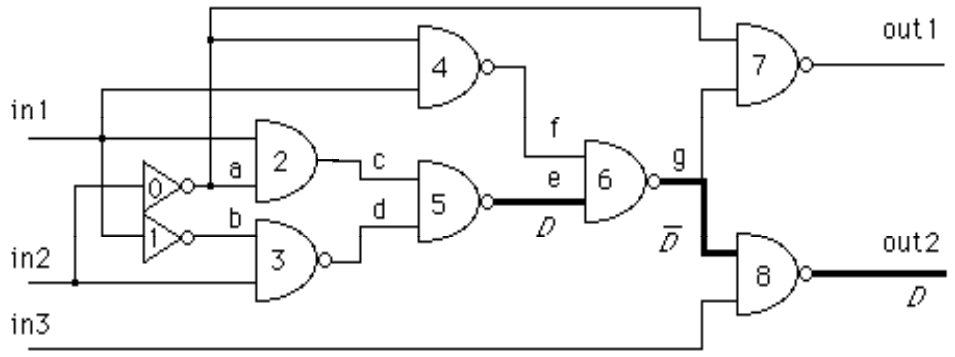
5-Variable D-Notation:

a\b	0	1	x	D	D*
0	1	1	1	1	1
1	1	0	x	D*	D
x	1	x	x	x	x
D	1	D*	x	D*	1
D*	1	D	x	1	D

27

You were introduced in B3 to the D Notation. The symbol D represents the value 1 in a good circuit, and the value 0 in a faulty circuit. The inverse of D, written here as D* has the opposite meaning, representing 0 in the good circuit and 1 in the faulty circuit.

The D-Algorithm - 1



To differentiate between a good and a faulty circuit it is necessary to activate the fault by choosing circuit inputs which ensure that the terminals of the faulty gate are set to the opposite value. This process is called *fault justification*. Then it is necessary to find a consistent set of circuit input values such that a *sensitised path* exists between the fault and a circuit output. Every node on this path can only have the value D or D*, since each node's value must be dependent on the presence or absence of the fault. This process is called *fault propagation.* Such a path is shown by the heavier lines in the figure.

The D-Algorithm - 2

- Draw up a target fault list
- Assign a D to one of the s-a-0 faults (D* to a s-a-1)
- Follow implications
- Justify this value & iterate until inputs
- Propagate fault effect & iterate until outputs
- If conflict, backtrack and change an earlier decision
- Iterate for all faults
- Minimise test set

29

To illustrate the operation of the D-algorithm, consider the generation of a test for Node-e SA0 on the previous slide. Initially all the node values in the circuit are unknown and thus are set to x. Having chosen the target fault, the first step is to assign a 1 to Node-e.

After each assignment a check is made to see if there are any *implications* for the other nodes. For example, assigning a 0 to one input of a NAND gate would imply that the output should take on the value 1 regardless of any other input values. If any inconsistencies arise, such as the implication of an assignment contradicting that of an earlier one, then the process needs to backtrack to the last arbitrary assignment made and alter it.

In this example there are no implications from assigning a 1 to Node-e and the next step is to *justify* the values assigned to nodes c and d. Justification and implication phases are carried out in turn until the target fault has been justified as far as the primary inputs.

Now it is possible to propagate the *fault effect* through Gate-6 by a suitable assignment of Node-f and the whole process is iterated until the fault effect is propagated to an output.

Reconvergent fan-out!

30

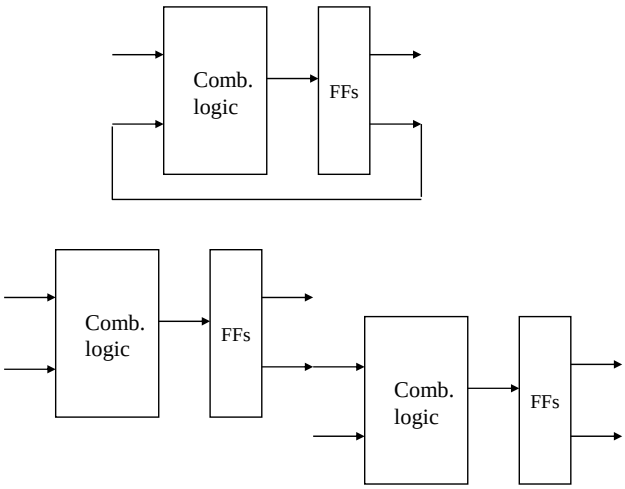
The situation can become particularly complex if the sensitised paths incorporate any reconvergent fanout, where different signal paths originating from the same component reconverge at one point further on in the circuit. See for example node g in the previous example.

Sometimes the requirements of the two paths are irreconcilable showing that the fault is uncontrollable. Sometimes the D values cancel each other showing that the fault is unobservable. These are both symptoms of redundant logic which might therefore be deleted, or else its testing disregarded. Other cases can require complicated backtracking and other ATPG algorithms have been developed to improve the situation.

ATPG can be VERY time consuming and may never reach 100%. The highest fault coverage may be obtained by a combination of ATPG and manual TPG.

Designers may be pleased to achieve 98% or 99% coverage. Fortunately this is rarely as bad as it sounds because the faults which are not tested may be very unlikely ones having marginal or even no effect on the circuit. For example, they may exist in circuits which are redundant (i.e. they have no actual affect on the overall circuit function) or they may arise where there is no legitimate failure mechanism that could cause them. If the fault coverage were to be weighted by the actual probability of occurrence the % will rise because lots of faults affect a bigger area than covered by the stuck-at model.

ATPG for Sequential Logic



31

Although test pattern generation for combinational logic is “more or less” a solved problem, test pattern generation for sequential logic is so much more complicated that it still represents a challenging problem.

Approaches are usually based on an “iterative” solution. The original circuit (assuming D-type flip flops) is replaced by multiple copies of the combinational logic, where each copy represents the original circuit at one instance of time. Outputs of the copy representing time i feed into the copy representing time $i + 1$ etc.. Tests can then be generated for the new (and larger!) circuit using the techniques above for combinational circuits.

Research has also been done on higher level approaches but has only been really successful in the case of datapath designs.

Fault Coverage & Fault Grading

- Toggle test
- Statistical fault grading
- Deterministic fault grading
  - output faults only
  - input and output faults

32

ATPG can be VERY time consuming and may never reach 100%. The highest fault coverage may be obtained by a combination of ATPG and manual TPG.

Designers may be pleased to achieve 98% or 99% coverage. Fortunately this is rarely as bad as it sounds because the faults which are not tested may be very unlikely ones having marginal or even no effect on the circuit. For example, they may exist in circuits which are redundant (i.e. they have no actual affect on the overall circuit function) or they may arise where there is no legitimate failure mechanism that could cause them. If the fault coverage were to be weighted by the actual probability of occurrence the % will rise because lots of faults affect a bigger area than covered by the stuck-at model.

Unless the test has been generated by an exhaustive procedure, the computation of the fault coverage requires a fault simulation of the circuit; i.e. for each possible fault the circuit is simulated and compared to the fault free simulation. This can be a *very* slow and expensive task (it may even take weeks of CPU time for a big IC!). It is possible to buy “hardware accelerators” to speed up the problem. For this reason, we may settle for an *estimate* of the fault coverage.

The SSA Model in Detail

- Output Faults
- Input Faults
- Gate Model

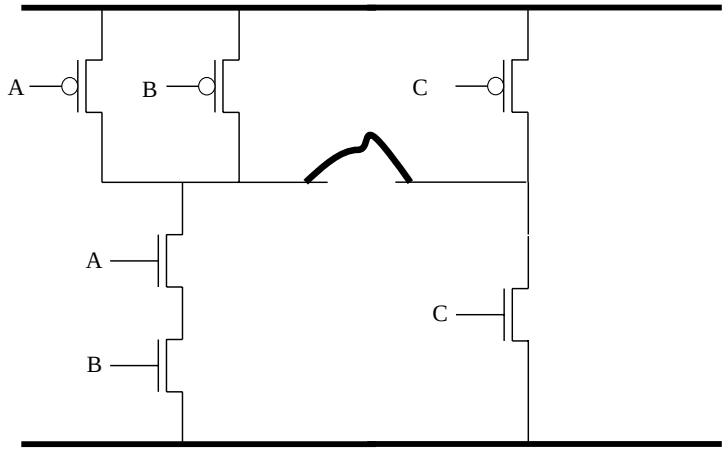
33

Despite its limitations (see following slides) the Single Stuck-at Fault Model is the primary fault model used in industry because it is the most tractable. There are however a few points to note in using it.

At fan-out nodes we could simply assume the driving output is stuck-at but it is MUCH BETTER to assume that each of the receiving inputs may also be separately stuck-at. That way you are sure to exercise every path through the circuit.

Likewise we could take rather large cells and therefore have a smaller number of nodes to test. It is again MUCH BETTER to consider only primitive cells (AND, NAND, OR, NOR - the number of inputs turns out not to matter) and avoid even seemingly innocuous things like ExORs.

Shorts (bridges)



34

With a bridge like this, there isn't a problem when the gates both produce ONE or both produce ZERO. But if the two gates produce conflicting values, one will usually win.

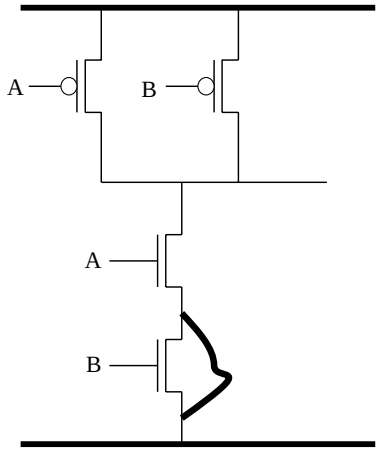
In this case, if we assume equal strength transistors, it looks likely that conflicting values will always result in a ONE. That's a pretty rash assumption however.

It is often helpful to categorise bridges as "wired-OR" as here, as "wired-AND" where conflicting values always result in a ZERO, or "dominant driver" where one gate always wins.

Nevertheless we nearly always start off with the stuck-at fault model. If a gate displays stuck-at-0 behaviour some of the time but not all the time we may suspect a wired-OR bridge . Likewise wired-AND & stuck-at-1. If a gate is sometimes s-a-0 and sometimes s-a-1, and sometime OK, we may suspect a dominant driver fault.

The real problem is the number of possible pairs of gates to check out. Often we will filter these by analysing our layout to see where signals are adjacent to each other. This is usually pretty successful in PCBs and often much more difficult on ICs.

Shorts (bridges)

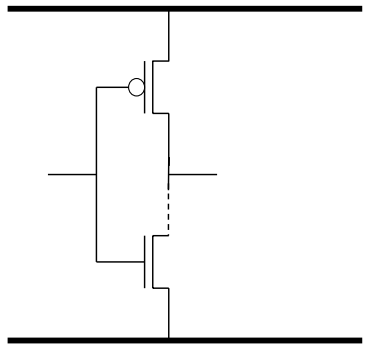


35

Some faults cannot be detected by their effect on the logic. In this example it depends on the resistance of the bridge and the transistor strengths. Although we might not notice it at the beginning, this kind of fault can cause intermittent faults or premature failure

Sometimes it is necessary and/or useful to use IDDQ testing – i.e. we measure the quiescent value of the power supply current IDD. In CMOS circuits this should be small when the signals aren't changing, but leakage currents are growing, which makes IDDQ testing more difficult.

Opens (breaks)



36

To test this fault, we need to make sure the output is high first and then try to pull it low – a “two-pattern” test.

Memories

- Pattern sensitivity
- Checkerboard Tests

37

Some popular circuit architectures require/lend themselves to particular styles of test. In memories, it is important to probe for potential charge sharing effects. Hence we normally want to test one cell at “1” while the adjacent cells are “0” , and *vice versa*.

PLAs

- Crosspoint faults

38

PLAs are best tested using “crosspoint” fault model which means testing to see that there is no connection where there should not be one and *vice versa*.

PCBs

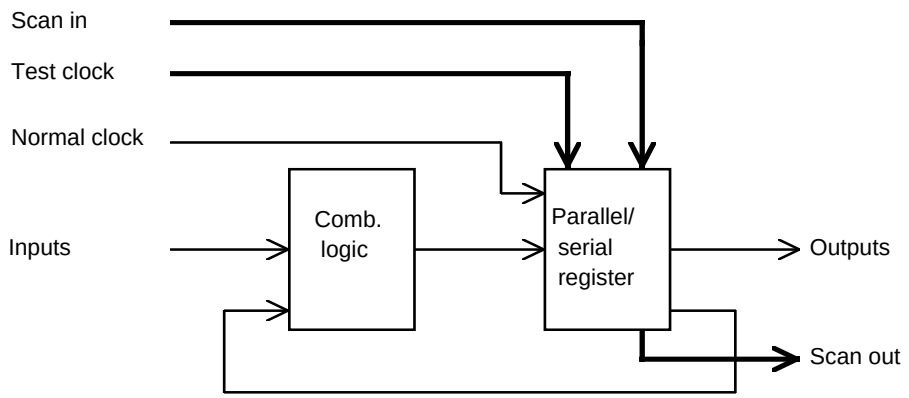
- Sorts
- Opens
- Pin-to-Pin Shorts
- Pin Opens

39

The majority of faults on bare Printed Circuit Boards consist of track-to-track shorts and open circuits. Usually PCBs are pre-tested and then populated with pre-tested components. At this stage the most likely faults are pin-to-pin solder shorts and pin opens.

Recall the problems you have experienced personally!

Design for test: Scan Path



40

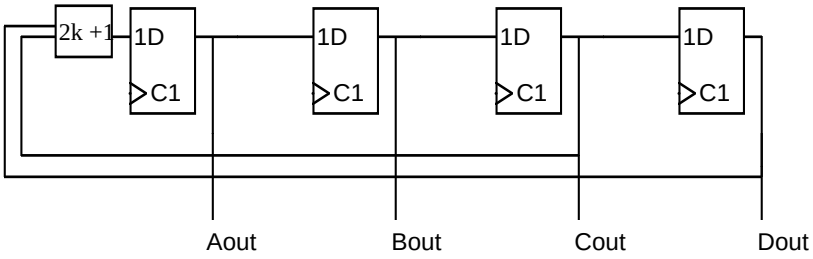
Since the problem of testing sequential circuits is so much more difficult than testing combinational circuits, one very effective approach is to put all the system flip-flops on a serial *scan path*. That way we can scan a test pattern onto the flip-flops, clock the system once, capture the results on the flips-flops and scan the result out.

This greatly simplifies the testing, but there is still a problem of the time taken to scan data in/out. We may try to reduce this by entering values which can be used for several cycles before scanning out again. This approach can be further improved by using a development called *scan set* where the scan chain register is separated from the functional circuit and can be loaded or read in parallel with the normal operation.

There can also be an impact on the circuit speed. (That is the reason why in the above diagram I have assumed separate parallel and serial mode clocks which we can implement with two different master sections to our flip-flops; the more obvious alternative of using multiplexers will put the multiplexer delay in series with the normal system logic).

There is obviously an appreciable overhead in terms of extra circuitry and routing, so we may try to get away with “partial scan”, using it only in the areas that are most difficult to test.

Built-in Test: Pseudo-random Testing

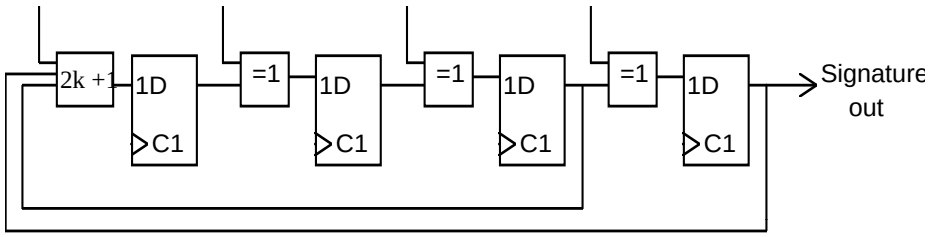


41

In order avoid the time taken to scan tests in, we can instead generate them inside the chip. A popular way of doing this is to use a "pseudo-random binary sequence generator" (PRBS) which we can easily make from a *n*-stage serial shift register with appropriate feedback taps (LFSR) to produce a sequence of 2^n - 1 patterns. This is an example which generates a sequence of 15 patterns. The required taps are well documented in the literature (e.g. in McCluskey).

Of course this approach means that we lose the ability to target specific faults as we have been suggesting with test pattern generation; instead we just blast the circuit with say a million patterns at random. Of course with up to 20 inputs, a million test patterns will give us exhaustive testing. It has been shown however that if a combinational circuit is fed by a number LFSRs of different lengths but each between 10 and 20 bits long, a million test patterns will almost always give us very good fault coverage.

Signature Analysis

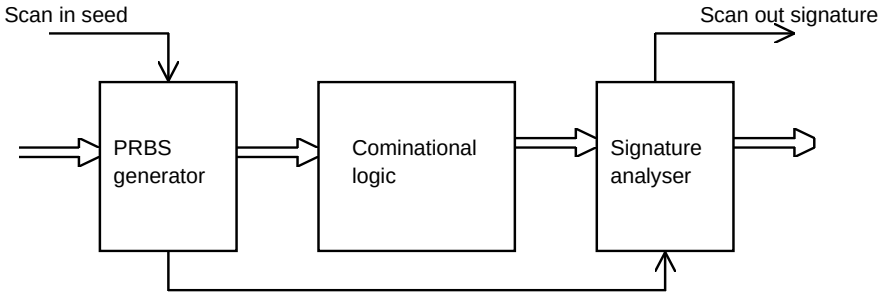


42

In a similar way we can use a pseudo-random sequence generator to compress consecutive results into the scan chain so that we only have to scan out the "signature" at the end of all the tests.

Signals applied to A, B, C, D cause "random" shifts to the sequence and hence if A, B, C, D are the points we wish to observe, the result at the end of the tests will be a unique "signature". For a long enough register (n > 15, say) the chances of multiple errors in A, B, C, D cancelling out become vanishingly small.

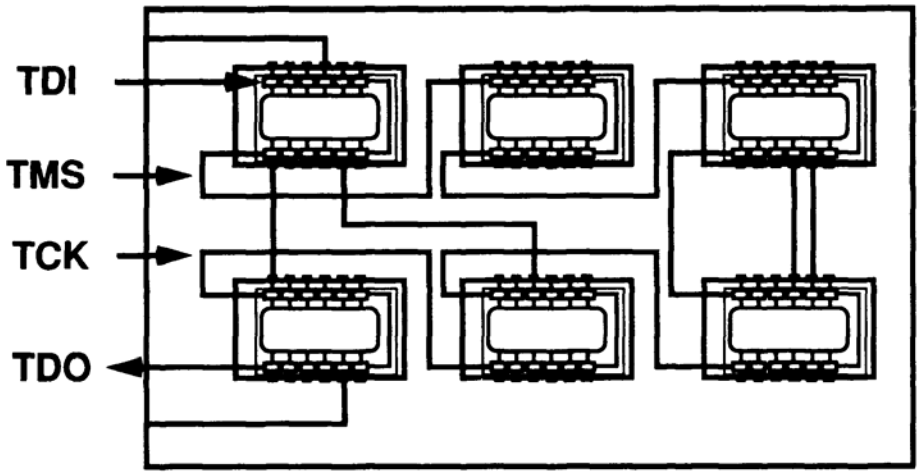
Built-in logic block observer (BILBO)



43

Finally we can put the PRBS generator and the SA register together in what is known as a BILBO configuration where our test consists merely of scanning in an initial value to seed the LFSR, clocking the system for a predetermined time and scanning out the final signature.

PCB Testing: Boundary Scan



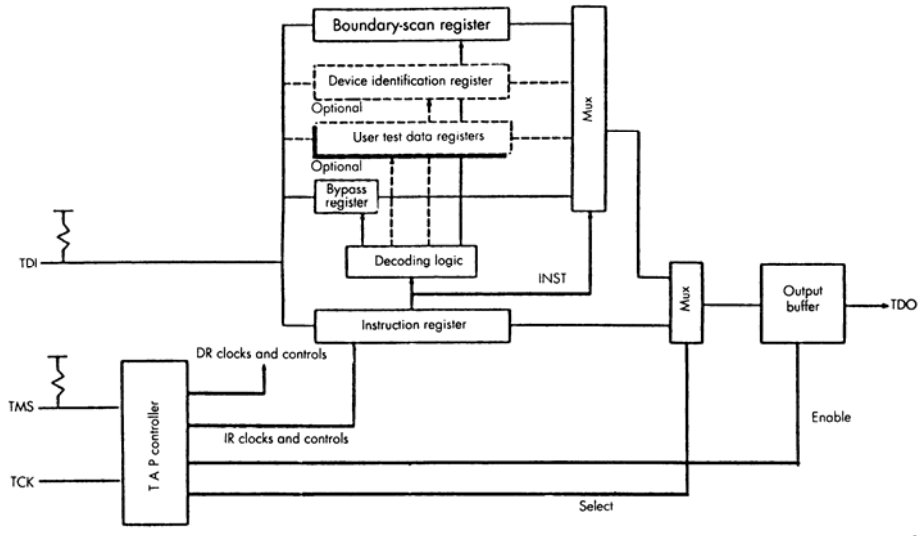
44

If the testing of ICs is difficult, then the testing of populated PCBs is potentially even more difficult. It is usual to test all the components before they are inserted and therefore our fault model is likely to be simplified to require only a check for liveness of the components and for continuity (opens and bridging shorts) of the copper tracking. With the development of surface mount technology and very fine tracks, it is increasingly difficult to perform “in-circuit testing” because the bed-of-nails tester is very expensive and likely to damage the tracks.

For this reason, the IEEE co-ordinated a Boundary Scan standard to encourage IC manufacturers to place a scan path around the periphery of their chips.

Where ICs do not have boundary scan and their testing is difficult, it is possible to insert scannable buffer chips.

Boundary Scan Control



Each chip is wired into a serial scan path from Test Data In (TDI) through to Test Data Out (TDO) and also receives a Test Mode Select (TMS) and Test Clock (TCK) input. The latter two signals are used to control the boundary scan path in a number of modes via an on-chip Test Access Port (TAP) controller.

Test modes include reading a device identification register which ensures we have inserted the correct ICs and scan testing of the the chip to chip interconnections. It is also possible to perform other tests such as scan testing of the ICs themselves, sampling the normal operating data and bypassing any given IC on the scan path.

Interconnect Testing

- Walking 1’s
- Test length
- Fault models

46

The classical test pattern generation for PCB interconnect diagnostic testing is a so-called walking-1s (0s) test where each line in turn is set to a 1 (0) with all the other lines set to 0 (1).

The problem of testing the interconnects is well known to board manufacturers but two additional factors are introduced with the advent of boundary scan.

Firstly, the time taken to apply a test via the boundary scan chain can be long and this puts a premium on reducing the number of test patterns. The walking 1’s test requires a number of tests equal to the number of interconnect paths. Therefore other algorithms can be used to reduce the number of vectors, either by performing on-line diagnostic test generation or else by taking the board layout into account .

Secondly, tests must be devised which take into account the fact that faults can only be detected by the effect they produce on the receiving gates, in contrast to "in-circuit" testers which can measure the actual fault current. In CMOS technology also we can no longer assume that bridging faults can be modelled as wired-AND (for TTL) or wired-OR (for ECL) and this needs to be taken into account in the algorithm.

Testing IP cores

- Scan paths
- BIST
- Wrappers

47

Buying in IP cores means that you may not know the details of the logic inside the cells and have to rely on the test patterns supplied.

Testing designs which already have structure.

- Memories
- Microprocessors
- Datapaths

48

Memories: the stuck-at fault model has been shown not to be adequate for testing memories which are susceptible to electrical problems such as *pattern sensitive faults*. Marching checkerboard patterns have been developed in response to this problem and can be generated very simply (and are amenable to BIST).

Microprocessors: direct observability of the microcode output is crucial to testing microprocessors in a realistic time and direct controllability of the control lines often allows the data path to be partitioned and accessed over the normal busses. Some microprocessors have self-testing programs built-in to their ROMs, though for contractual reasons these have often not been made available to the customer.

Datapath testing.

- Easy propagation/justification
- Pseudo-exhaustive tests
- C-testability
- Linear-testability

49

Systolic, regular array, pipelined & datapath architectures: it is usually possible to propagate regular test patterns through such architectures in a similar way to the normal data propagation and hence to produce exceptionally short test sets. (If you can work out how to test one "bit-slice", then you may well be able to apply tests in parallel to all the bit slices - that's how I happen to know that an *n*-bit adder can be "pseudo-exhaustively" tested in just 8 tests - a bit less than 2^{2n}!)

Moral: if the design possesses a distinct structure, exploit it for test; if it is unstructured, impose structure upon it!

Delay-fault testing

- Transition fault
- Path delay fault
- Two pattern tests
- Slow-fast-slow scan

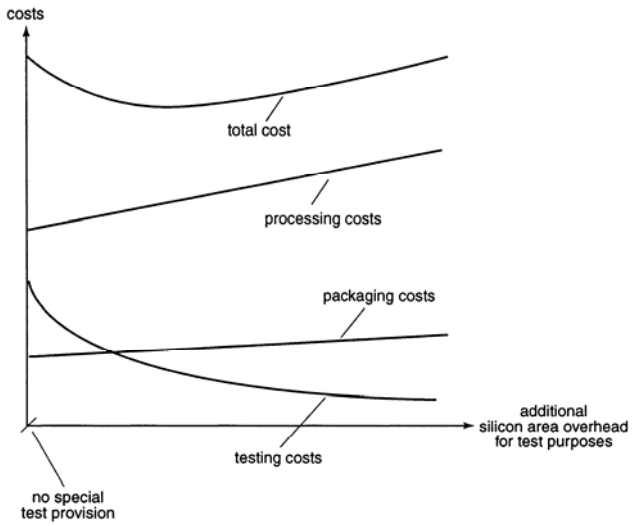
50

A “transition fault” is a slow-to rise or slow-to-fall at one particular gate.

A “path delay fault” is a slow path through a number of gates.

The problems with two-pattern tests are (a) they may be invalidated by hazard activity and (b) your tester probably and/or your scan probably paths won’t go fast enough. In the slow-fast-slow method the first test vector is scanned in slowly, then there is one very fast clock transition and then you can scan out the result slowly again

The Cost of Testing



51

It has recently been suggested that testing often accounts for more than half the cost of a design. You may be able to corroborate that from your own experience? Therefore there is a very real incentive to design a circuit to be as testable as possible.

Detection vs. Diagnosis

- Most of my remarks are directed to merely detecting faults; depending on the circumstances, we may also want to diagnose the fault.
- When we want to diagnose a fault, it is common to employ either, or both of:
  - Cause-effect diagnosis
  - Effect-cause diagnosis

52

Cause-effect diagnosis means we assume that some fault exists, work out what it would do and compare that with what we see in the real thing.

Effect-cause diagnosis means we trace back from the failing outputs to see which gates could be implicated. By examining the intersection of the resulting “logic cones”, we hope to be able to pin down the problem.

Both approaches can be extraordinarily complicated and of course may use pretty sophisticated (and time-consuming) software. In real life we may have more than one fault and none of them may display the clear-cut symptoms of our fault model. You may therefore find it pays off (e.g. in your project work), to assume you won’t solve the problem every time and go for the things which are more likely to pay off – try effect-cause on the most likely faults & effect-cause to identify the most likely regions of your design.