CHAPTER 02

DIGITAL CIRCUIT TESTING
CHAPTER 02

DIGITAL CIRCUIT TESTING

If you cannot measure it, you cannot control it.
If you cannot control it, you cannot manage it.
If you cannot manage it, you cannot improve it.

Dr. H. James Harrinton, 1991.

2.1 Test and Measurement, What and Why?

Why the measurement is important? Beyond supplying the vocational and professional needs, why does the measurement merit further examination?

Measurement is so universal as not to merit a second thought. Yet, without it, the exchange of goods and services cannot take place. It is always necessary to human society as language, and it occurs in the early civilizations, long before the technology, and these current days. Without the conscious decision, to agree on a way of measuring, cooperative activity could hardly take place. With it, marketplaces and increasing sophisticated economics can develop, matching barter, cash or credit to whatever is owned by one and desired by another. [6].

Measurement is made to reveal a state or current condition and possibly to notify and alert the designer or user. Things are measured to quantify the magnitude of phenomenon. Probably, most importantly, in order to control process. All these should lead to insight to phenomena and processes, as the ultimate aim of the measurement is to control mange and improve. Ultimately it provides an interpretation of events and current status of the device or system under test.

Measurement provides definition and helps to understand quality and consistency. Quality can't be replicated, if it can not be measured at first. Manufacturing requires measurement to replicate merchandise and control its production. Measurement also helps to express appreciation. For example, if video system and photoelectric sensors are not there, it couldn't always possible to settle a horse race without degenerating into tribal warfare. [7] It can be concluded that the measurement is a science, which is integral to life. It follows then that, if measurement is needed, instruments and test system will be needed, which may be
hardware, may be software or may be mix of two. As the quality of a measurement is desired to be improved, so that of the test and measurement systems.

Whenever a phenomenon occurs, people set about to understand it. First, it is tried to recognise as a phenomenon, then one tries to describe it qualitatively. After a familiarity, one can begin measure it quantitatively. With time, refinements are made to measurement and standards are set in due course. Finally, a measurement is incorporated into regular processes that are used for advancement of capabilities. The recognition that accuracy allowed comparison, in turn, that underlined the need of standard, which is a component to the genesis of the several international standards bodies that exist today.

• **Basic principles and giving meaning to measurement:**

  Every measurement must have a meaning. It represents a substantial result with which the person who performs the measurement can relate. It provides a relationship of one domain to another. Measurement is always on the basis of a strong reference; a relationship between the domains can be established. [8]

  In most important part of giving meaning to measurement is to provide understanding of process. Understanding should extend throughout the process including the physical phenomenon involved, actual sampling and measurement, its transformation and data manipulation. It also involves result generation for further action or decision-making. The components of measurement understanding includes few key-points mentioned below. [7, 8]

  **Visibility:** A measurement made must provide visibility into basic process of system, parameters of which are under measurement. So that the user should, receive a recognisable interpretation of data.

  **Clarity and hidden complexity:** A good understanding of measurement of process parameter should provide simplicity and clarity, not the ambiguity. It also should be taken care that what is the attribute of interest, that should be emphasized, not what the measurement system is capable to perform. Failing to this, the measurement itself will be a complex task, for no reasons of need as well achievements.

  **Traceability to a reference:** Measurement results are meaningless, if there is no basis for its reference and Traceability to standards. There are quite nos. of stories in history that the researches are failed due to not caring about the references and Traceability. If the results are not traceable, it is just business of collecting high-resolution, meaningless data. [8]
2.2 Spectrum of Test Engineering

It is amazing to know, the spectrum of Test and measurement engineering, as it doesn’t only encompasses electronics, but the entire engineering and technology, life sciences, chemical and entire spectrum of scientific applications. Electronics is just the way to do it. Applications are widespread in all areas of life and business as well technology itself. Revenue generated of a single major giant, in the field of Measurement and Instrumentation is above $05 Billions for the year 2005-06, [9]. One may not able to say off the hand, how many zeroes will follow that 05. This is just one for example,

How to measure a technologies effect on a Community? When the field is measurement and the Community is global, the results may not be measurable by precision tools, but by the impact they make, on people’s lives. Testing and Measurement is an integral part of life. It starts with the record of birth date and time, and follow one up to the last moment of life, even that is recorded and analysed for further improvement of human civilization.

Today’s consumer and business electronics products are small, complex, and powerful. Leadership of electronic measurement helps enable the creation, reliability, and advancement of these devices. The premier measurement companies, providing the critical tools and technologies that sense, measure, and interpret the physical and biological world. Innovative technology solutions enable a wide range of customers in communications, electronics, life sciences, and chemical analysis to make technological advancements that drive productivity and improve the way people live and work.

Medical and life science development is tremendous, in last few decades, but if we see the contributions, major share goes to tests and measurement solutions. Majority of the tests are conducted electronically. Advancement in the processor technology, which enables development of high-end measurement and diagnostic tools, has a great share in the field. A special branch of engineering working behind this is biomedical engineering, which deals with medical science aspect, in order to design and develop high-end equipments to meet the current demand.
2.3 Testing of Digital Circuits

- **The Test:**

  In its most general sense, can be viewed as an experiment whose purpose is to confirm or refute a hypothesis or to distinguish between two or more hypotheses. Things don't always work as intended. Some devices are manufactured incorrectly; others break or wear out after extensive use. In order to determine if a device was manufactured correctly, or if it continues to function as intended, it must be tested.

  The test is an evaluation based on a set of requirements. Depending on the complexity of the product, the test may be a mere perusal of the product to determine whether it suits one's personal whims, or it could be a long, exhaustive checkout of a complex system to ensure compliance with many performance and safety criteria. Emphasis may be on speed of performance, accuracy, or reliability. [11]

  Quality frequently surfaces as a topic for discussion in trade journals and periodicals. However, it is seldom defined. Rather, it is assumed that the target audience understands the intended meaning in some intuitive way. Unfortunately, intuition can lead to ambiguity or confusion.

  Both, the cost of developing tests and the cost of applying tests to individual units will be considered. In some cases it becomes necessary to make trade-offs. For example, some algorithms for testing memories are easy to create; a computer program to generate test vectors can be written in less than 12 hours. However, the set of test vectors thus created may require several millennia to apply to an actual device. Such a test is of no practical value. It becomes necessary to invest more effort into initially creating a test in order to reduce the cost of applying it to individual units.

**2.3.1 Few Terms and Definitions**

Before introduction of basic concepts of testing of digital integrated circuits, let us define a few terms that will be used frequently throughout this thesis. A Boolean variable is an entity, which can be assigned the value of either zero or one. A one dimensional array of Boolean variables that varies as a function of time is represented as \( a(t) \). For a single time frame, we represent the Boolean function as \( a(t_0) \).
The system model used here is a gate level description of a synchronous Sequential circuit, shown in Figure 2.1. The sequential circuit contains logic gates and memory elements, where all memory elements can be initialized to a known value in a single time frame. The sequential circuit has \( m \) primary inputs (PIs), \( x(t) \), \( p \) primary outputs (POs), \( z(t) \) and \( n \) memory elements or flip-flops. The pseudo-primary inputs (PPIs) of the circuit are the inputs of the combinational logic gates that are connected to the outputs of the flip-flops. The pseudo-primary outputs (PPOs) of the circuit are the outputs of the combinational logic gates that are connected to the inputs of the flip-flops.

All wires of the sequential circuit are given a unique label. The wire of interest, was shown in Figure 2.1 is one which models a fault by associating with it a constant value of 0 or 1. A stuck-at fault on wire \( w \) can be represented as \( w/0 \) or \( w/1 \) denoting wire \( w \) stuck-at zero and one respectively. The Boolean function on wire \( w \) is denoted by \( f_w \). If \( w \) labels a wire which is a fan-out stem, then the Boolean function on all wires which branch from \( w \) are denoted as \( f_w \). The output of the good machine, or the output of the fault-free sequential Circuit, \( z(t) \), is defined as:

\[
Z(t) \xrightarrow{\Delta} \Lambda(x(t), y(t), f_w(x(t), y(t)))
Y(t) \xrightarrow{\Delta} \Delta(x(t), y(t), f_w(x(t), y(t))
\]

Where \( y(t+1)=y(t) \).

The \( y(t) \) is the present state and the \( Y(t) \) is the next state of the sequential circuit. The sequential circuit is characterized by the next state function \( \Delta \), the output function \( \Lambda \) and the function at wire \( w \), \( f_w \). Sequential circuit model is a Mealy machine, because the output depends on both the primary inputs and the present state of the machine at time \( t \), Any Moore
machine, where the output is a function of the present state only, can be represented by an equivalent Mealy machine with the same input-output characteristics, so the model is general.

**Definition 1 (Faulty Machine):** The faulty machine with respect to the single stuck-at fault \( \gamma \) on wire \( w \), where \( \gamma = \frac{w}{0} \) is based on the good machine with \( f_w(t) = 0 \) or

\[
\mathbf{z}_{w,0}(t) \overset{\triangle}{=} \Lambda(x(t), y_{w,0}(t), f_w(t))|_{f_w(t)=0} \\
= \Lambda(x(t), y_{w,0}(t), 0)
\]

\[
\mathbf{Y}_{w,0}(t) \overset{\triangle}{=} \Delta(x(t), y_{w,0}(t), f_w(t))|_{f_w(t)=0} \\
= \Delta(x(t), y_{w,0}(t), 0)
\]

where \( f_w(t) \) is used as a shorthand notation for \( f_w(x(t), y_{w,0}(t)) \). Similarly, we can denote the faulty machine with respect to the fault \( \gamma = \frac{w}{1} \) as \( Z_{w,1} \). The faulty machine is the model of the good machine with the single stuck at fault \( \gamma \) on wire \( w \) included. The outputs and the state of the faulty machine \( Z(t) \) and \( Y(t) \) may not be the same as that of the good machine \( Z(t) \) and \( Y(t) \) even if both machines are initialized to the same state.

**Definition 2 (Controllability Functions of a Single Wire):** Given a wire \( w \), the controllability functions are simply \( f_w[x(t)] \) and \( f_w[y(t)] \) both are functions of primary and pseudo primary input variables \( x(t) \) and \( y(t) \). A wire is not controllable if either \( f_w(t) = 1 \) or if \( f_w(t) = 1 \) for all \( x(t) \), where \( t \) is an integer greater than or equal to zero. The controllability functions can be used to determine whether a wire \( w \) can be set to the value of one or zero. For a given sequential circuit which has been initialized to a known state, the wire \( w \) is said to be controllable to one if there exists an \( x(t) \) such that \( f_w(t) = 1 \). Similarly, the wire \( w \) is said to be controllable to zero if there exists an \( x(t) \) such that \( f_w(t) = 1 \). A wire is fully controllable, if it can be controlled to both one and zero.
Definition 3 (Observability Function of a Single Wire): Given a wire w, the observability function with respect to the r\(^{th}\) output:

\[
O_r^w(t) \triangleq \Lambda_r(x(t), y(t), \overline{f_w(t)}) \oplus \Lambda_r(x(t), y(t), f_w(t)).
\]

---

evaluating to the familiar Boolean difference of respective cofactors:

\[
O_r^w(t) = \Lambda_r(x(t), y(t), \overline{f_w(t)}) |_{f_w(t)\neq 0} \oplus \Lambda_r(x(t), y(t), f_w(t)) |_{f_w(t)\neq 1}
\]

---

With respect to all outputs, the observability of wire w is:

\[
\mathcal{O}_w(t) \triangleq \sum_r O_r^w(t).
\]

---

where \(\sum_r\) designates OR-ing of observabilities from all outputs. A wire is not observable if \(O_w(t)=0\) for all \(x(t)\).

The observability function can be used to determine whether the defects of a fault on wire w can be observed at one or more primary outputs. For a given sequential circuit which has been initialized to a known state, a fault on wire w is said to be observable at one or more primary outputs at time t if there exists an \(x(t)\) such that \(O_w(t)=1\). When the fault is observable and \(x(t)\) is applied to both the good and faulty machines, beginning from the same initial state, the output of the two machines is different.

Definition 4 (Detectability Functions of a Single Wire): Given a wire w, the detectability function with respect to the r\(^{th}\) output:

\[
\tau_r^w(t) \triangleq A_r(x(t), y(t), f_w(t)) \oplus A_r(x(t), y(t), \overline{f_w(t)}) |_{f_w(t)=0}
\]

---

Evaluating, respectively, to:

\[
\tau_{w/0}^r(t) = f_w(t).\mathcal{O}_w(t)
\]

\[
\tau_{w/1}^r(t) = \overline{f_w(t)}.\mathcal{O}_w(t)
\]

---

With respect to all outputs, the detectabilities are:
The detectability function can be used to determine if a test for a fault $\gamma$ on wire $w$ exists. For a given sequential circuit which has been initialized to a known state, a fault on wire $w$ is said to be detectable (i.e., a test for the fault exists) if there exists an $x(t)$ which satisfies the above equations i.e. $r_\gamma(t)=1$. A fault $\gamma$ is detectable if the good machine and the faulty machines have different outputs when the same inputs have been applied to both machines and both were initialized to the same state. The fault $\gamma=w/0$ is detectable if the fault on wire $w$ is controllable to one and observable at one or more primary outputs under the same input conditions. Similarly, the fault $\gamma=w/1$ is detectable if the fault on wire $w$ is controllable to zero and observable at one or more primary outputs under the same input conditions.

**Definition 5 (Test Sequence):** A test sequence for the fault $w/0$ is an ordered sequence of primary input assignments also called input stimulus, $X(t)$, where $t$ is an integer greater than or equal to zero, which detects the fault or satisfies the detectability function $\tau_{w/0}(t)=1$, in time frame $t$.

Similarly, a test sequence is defined for the fault $w/1$ as $x(t)$, where $t$ is an integer greater than or equal to zero, such that $\tau_{w/1}(t)=1$, in time frame $t$.

A test sequence is input stimulus which when applied to a sequential circuit detects at least one fault $\gamma$. The test sequence has length $k$ if there exists an $x(t)$, such that the detectability function is satisfied at time $t=k$. A test sequence is said to be tentative if the sequence has yet to be shown to detect a fault.

**Definition 6 (Fault Simulation Experiment):** Given a sequential circuit, initialized to a known state at time $t=0$ and a tentative test sequence, $x(t)$, $t=0,1,\ldots,k$, we define the fault simulation experiment for fault $w/0$ as the evaluation of the detectability function
Tw/0(t). If for Tw/0(t)=1 for t <= k, the fault w/0 has been detected and the test sequence is no longer tentative.

Similarly, we can define the fault simulation experiment for fault w/1 by evaluating the detectability function Tw/1/(t)=1, for the given tentative test sequence.

Figure 2.2 Fault Simulation Experiment.

Figure 2.2 illustrates a fault simulation experiment for the fault w/0. At time, t =0, the good machine and the faulty machine are initialized to the same state and can be represented as:

\[ Z(t) = A\{x(t) - y(t), f_{w_0}(t)\} \]  \hspace{1cm} (2.12)

\[ = A\{x(t), y_0(t), y_1(0), \ldots, y_{n-1}(0), f_{w_0}(t)\} \]  \hspace{1cm} (2.13)

\[ Z_{w_0/0}(t) = \Lambda\{x(t), y_{w_0/0}(t), f_{w_0}(t)\} \]  \hspace{1cm} (2.14)

\[ = \Lambda\{x(t), y_0(t), y_1(0), \ldots, y_{n-1}(0), 0\} \]  \hspace{1cm} (2.15)

Even though both machines are given the same input stimulus, at time t>0, the state of the good machine, y(t), is not necessarily the same as the state of the faulty machine. Some of the state bits, yi(t), of the good machine may have values which are the same in the faulty machine, while others yj(t), in the good machine, may have complementary values in the faulty machine.

Definition 7 (Test Generation Problem): Given a fault w/0 in a sequential circuit which is in a known state at time t =0, the test generation problem is defined as the task of finding a test sequence which detects the fault. The solution is a test sequence for x(t), t >=
which satisfies $Tw/0/(t)=1$ thus detecting the fault w/0. The test generation problem for a fault w/1 can be defined similarly.

A test sequence may or may not exist for an arbitrary fault $\gamma$. Algorithms for sequential test generation which can prove that a test does not exist for a given fault are said to be complete.

Test generation for a single time frame is equivalent to the well-known combinational test generation problem and is a special case of the sequential test generation problem. Assume that we are given a sequential circuit and the fault w/0 in a single time frame. We are to solve for $x(t_0)$ and $y(t_0)$, given the detectability function for a single time frame $Tw/0/(t)=1$. Here, the observability function, $Ow(t_0)$, is defined in terms of all primary outputs and pseudo-primary outputs and $fw(t_0)$ is controllable by both the primary inputs and pseudo-primary inputs.

**Definition 8 (Test Vector):** A test vector is an assignment of the primary inputs, $x(t_0)$, and pseudo-primary inputs $y(t_0)$ that satisfy the detectability function in a single time frame $t_0$. A test vector, when applied to the primary inputs and pseudo-primary inputs of a combinational circuit, detects at least one fault $\gamma$. A test vector is produced as a result of combinational test generation just as a test sequence is produced as a result of sequential test generation.

**Definition 9 (Sequentially Redundant Fault):** A fault w/0 (w/1) is sequentially redundant if no test sequence $x(t)$ exists which satisfies $Tw/0/(t)=1$ ($Tw/0/(t)=1$).

If a fault is sequentially redundant, the good machine and the faulty machine will always have the same output when both machines are initialized to the same state and both machines are presented with the same input stimulus. Upon identification of sequentially redundant faults, optimizations can be performed to reduce the size of the circuit.

Redundant faults may intentionally be placed in a circuit to improve circuit performance or to be able to detect errors which occur during the on-line operation of the circuit. Several high speed adders have implementations which include redundant faults, but in this thesis, we prefer an implementation which is irredundant. In the design of self-checking/correcting circuits, redundancy is introduced at the module level. Redundant modules are placed in parallel and a voting circuit places the output of the majority of the modules on the outputs of the system. While detection/correction of errors which occur while the circuit is on-line is an area of considerable interest, the emphasis of this thesis is off-line testing.
Definition 10 (Combinationally Redundant Fault): A fault on wire \( w \) is combinationally redundant if the detectability function can never be satisfied for the special case of the test generation problem in a single time frame. No test vector exists for the fault. This is a special case of the sequentially redundant fault for a single time frame.

When no test exists for a fault \( \gamma \) when considering the combinational test generation problem, the fault is combinationally redundant. Similar to sequentially redundant faults, combinationally redundant faults may be intentionally included in a circuit. Combinationally redundant faults always become sequentially redundant when the combinational circuit is embedded in a larger sequential circuit.

Definition 11 (Testable Fault): A fault on wire \( w \) is testable, if a solution of the test generation problem exists.

Given a testable fault \( \gamma \) in a sequential circuit, there exists a test sequence which detects the fault. Similarly, for a testable fault \( \gamma \) in a combinational circuit, there exists a test vector which detects the fault.

Definition 12 (Aborted Fault): A fault is declared aborted if we terminate solving the test generation problem due to a backtrack limit. Aborted faults are neither proven redundant nor proven testable.

The process of test generation is complex in terms of the amount of computation required. Several test generation algorithms allow the user to specify a limit on the amount of time which will be spent on test generation for a given fault \( \gamma \). If this limit, also called the backtrack limit is exceeded, then the program aborts while neither generating a test nor proving that the fault is redundant. In this case, the fault is termed an aborted or untestable fault.

- Testability:

  Testability goes by different definitions. Few papers and researches [12] defines testability as visibility and control. Visibility is a ability to observe the states, outputs, resource usage and other side effects of the system/software under test. Control is an ability to apply inputs to the system/software under test or place it in specified states. These are the aspects of testability that are affected by the system/software design and that have greatest impact on the feasibility of automated testing. Ultimately, testability means having reliable and convenient interfaces to drive the execution and verification of tests. [12]. Many papers and researches have proved [8,10, 12] that, humans are really bad at boring, repetitive tasks of testing the same systems for the whole time of work. If a test plan is based on the idea that
the staff will faithfully execute a long list of printed instructions and follow the procedure, at least once per release, then the testing is probably not effective. For example, many manual test plans contain long sequences of things the operator is required to do, often with information on the screen to be confirmed as correct. This is all very well for successful tests, but what happens when one fails? Usually, these test scripts cover large numbers of behaviors. There is thus a motivation to complete the rest of the script, rather than stop, which ultimately hampers on the effectiveness of testing and leads to need of automatic test equipments / Systems (ATE), which is addressed later in the text followed.

2.3.2 Evolution history and overview

The eighties was the decade where test problems were solved as an after-thought by involving massive test engineering to compensate for chip designers’ lack of knowledge in how to handle the test and quality issues of a given chip function. The design process itself typically was handled in a sequential manner where hardware design basically had to be completed before software. Design started, and then eventually test engineering started.

In the industry, a general consensus existed since the beginning of the nineties that the existing approaches of isolated hardware and software design paths were less than optimal. To bridge this, new approaches in digital design engineering like hardware-software co-design /co-verification, design reuse have emerged over the decade. This has made it possible for the design community to meet the rapid escalating complexity of chips and still keep design resource employment almost constant or even reducing the total design time.

As the IC industry matures, and engineers gain a better understanding of the many factors that contribute to yield loss, they are able to apply this new-found knowledge to reduce both the sizes and the numbers of defects that occur in a given die area, with the result that yields increase. This is all the more remarkable in view of the fact that feature sizes continue to shrink and chip complexity continues to increase. A relationship between complexity and minimum defect size is suggested in Figure 2.3, where trends are projected to the year 2010.
The incentive to shrink die size is motivated by a rather basic imperative, improved profitability. Consider a wafer with \( N \) die and a yield \( Y \). There will be \( Y \times N \) good die on the wafer. Each of these will be sold for \( Z \) dollars, producing an income of \( Y \times N \times Z \). This income must exceed the cost of designing, manufacturing, packaging, testing, and marketing the chips. If die size is reduced, there will be more die on each wafer, but the number of bad die may increase. If shrinking the die size causes a disproportionately larger increase in the number of good die, then income increases, assuming production costs do not go up disproportionately. Given a fixed selling price, then, the object is to find die size and yield that maximize the product term \( Y \times N \times Z \). A simplistic analysis could lead to the conclusion that the number of good die must increase disproportionately. Consider the following: If there were simply a fixed number of point defects on a wafer, and they caused \( (1 - Y) \) die to fail, then doubling the number of die on a wafer would produce \( N + (1 - Y) \times N \) good die. In effect, the overall yield increases.

- **Test Economics:**

  Test related costs for ICs and PCBs include both time and resource. For some products the failure to reach a market window early in the life cycle of the product can cause significant loss of revenue and may in fact be fatal to the future of the product. The dependency table in Figure 2.4 shows test cost broken down into four categories, some of which are one-time, non recurring costs whereas others are recurring costs. Test preparation includes costs related to development of the test program(s) as well as some potential costs incurred during design of the design-for-test (DFT) features. DFT-related costs are directed
toward improving access to the basic functionality of the design in order to simplify the creation of test programs. [11]

<table>
<thead>
<tr>
<th>Test preparation</th>
<th>Test generation</th>
<th>Tester program</th>
<th>DFT design</th>
<th>Test execution</th>
<th>Hardware</th>
<th>Tester</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test related silicon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imperfect test quality</td>
<td>Escape</td>
<td>Lost performance</td>
<td>Lost yield</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.4** Cost Break-up for testing a system / product

### 2.3.3 Basic Test Types and conventional test setup:

In its most general sense, a test can be viewed as an experiment whose purpose is to confirm or refute a hypothesis or to distinguish between two or more hypotheses. Figure 2.5 depicts a test configuration in which stimuli are applied to a device under-test (DUT), and the response is evaluated. If one knows, what the *expected Response* is from the correctly operating device, one can compare it to the response of the DUT to determine if the DUT is responding correctly. When the DUT is a digital logic device, the stimuli are called *test patterns* or *test vectors*. In this context a *vector* is an ordered *n*-tuple: each bit of the vector is applied to a specific input pin of the DUT. The expected or predicted outcome is usually observed at output pins of the device, although some test configurations permit monitoring of test points within the circuit that are not normally accessible during operation. A tester captures the response at the output pins and compares that response to the expected response determined by applying the stimuli to a known good device and recording the response, or by creating a *model* of the circuit (i.e., a representation or abstraction of selected features of the system) and simulating the input stimuli by means of that model. If the DUT response differs from the expected response, then an *error* is said to have occurred. The error results from a
defect in the circuit. The next step in the process depends on the type of test that is to be applied. [11]

**Figure: 2.5 Conventional Test Configurations**

Taxonomy of test types is shown in Table 2.1. The classifications range from testing die on a bare wafer to tests developed by the designer to verify that the design is correct. In a typical manufacturing environment, where tests are applied to die on a wafer, the most likely response to a failure indication is to halt the test immediately and discard the failing part. This is commonly referred to as a go-no go test. The object is to identify failing parts as quickly as possible in order to reduce the amount of time spent on the tester. If several functional test programs were developed for the part, a common practice is to arrange them so that the most effective test program—that is, the one that uncovers the most defective parts—is run first. Ranking the effectiveness of the test programs can be done through the use of a fault simulator. The die, that pass the wafer test are packaged and then retested. Bonding a chip to a package has the potential to introduce additional defects into the process, and these must be identified.

There are three major aspects of the test problem:

1. Specification of test stimuli
2. Determination of correct response
3. Evaluation of the effectiveness of the stimuli

Furthermore, this approach to testing can be used both to detect the presence of faults and to distinguish between several faults for repair purposes. In digital logic, the three phases of the test process listed above are referred to as test pattern generation, logic simulation, and fault simulation.
<table>
<thead>
<tr>
<th>Type of Test</th>
<th>Purpose of Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production</td>
<td>Test of manufactured parts to sort out those that are faulty</td>
</tr>
<tr>
<td>Wafer Sort or Probe</td>
<td>Test of each die on the wafer.</td>
</tr>
<tr>
<td>Final or Package</td>
<td>Test of packaged chips and separation into bins (military, commercial, industrial).</td>
</tr>
<tr>
<td>Acceptance</td>
<td>Test to demonstrate the degree of compliance of a device with purchaser’s requirements.</td>
</tr>
<tr>
<td>Sample</td>
<td>Test of some but not all parts.</td>
</tr>
<tr>
<td>Go-no-Go</td>
<td>Test to determine whether device meets specifications.</td>
</tr>
<tr>
<td>Characterization or Engineering</td>
<td>Test to determine actual values of AC and DC parameters and the interaction of parameters. Used to set final specifications and to identify areas to improve process to increase yield.</td>
</tr>
<tr>
<td>Stress screening (burn-in)</td>
<td>Test with stress (high temperature, temperature cycling, vibration, etc.) applied to eliminate short life parts.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Test after subjecting the part to extended high temperature to estimate time to failure in normal operation.</td>
</tr>
<tr>
<td>Diagnostic (repair)</td>
<td>Test to locate failure site on failed part</td>
</tr>
<tr>
<td>Quality</td>
<td>Test by quality assurance department of a sample of each lot of manufactured parts. More stringent than final test.</td>
</tr>
<tr>
<td>On-line or checking</td>
<td>On-line testing to detect errors during system operation.</td>
</tr>
<tr>
<td>Design verification</td>
<td>Verify the correctness of a design.</td>
</tr>
</tbody>
</table>
2.3.4 Manual Test Generation & response evaluation

A system is tested by applying a set of test pattern (vectors/stimuli) on its primary inputs and then compare the test response on its primary outputs with know good vectors. An illustration in Figure 2.6 shows a test control unit which controls the test pattern generator and the test response evaluator. Traditionally the test patterns are supplied from an external tester. However, due to the increasing capacity of the integrated circuit technology, a complete system consisting of several complex blocks can be integrated on a single chip. One of the advantages of this integration is that the performance can increase mainly because there is no chip-to-chip connection which used to be a major performance bottle-neck. Due to the increasing performance of systems and the limitation of bandwidth when using external testers, there is a trend in moving the main functions of the external tester onto the chip. This would mean that all blocks in Figure 2.6 are placed on chip.

![Figure 2.6 Test generation and Control (Single Unit)](image)

Furthermore, for large systems, it is not feasible to have only one test pattern generator and one test response evaluator as in Figure 2.6. An example of a system with several test pattern generators and test response evaluator is given in Figure 2.7. The test generators are often of different types with their own advantages and disadvantages. For instance, Test Pattern Generator 1 (TPG1) and Test Pattern Generator 2 (TPG2) can be of different types in order to fit respectively circuit-under test. One approach to minimizing test application time while keeping test quality high (fault coverage) is to allow flexibility where each circuit under test is to be tested by several test sets from different test generators.
2.3.5 Introduction to ATE

Issues related to On-Line Testing (OLT) are increasingly becoming important in modern electronic systems. These needs have increased dramatically in recent times with the widespread usage of deep submicron (DSM) technology, since there is a rise in the probability of development of faults during operation. Therefore OLT is an area of recent research and several techniques have been developed. On-line Testing techniques for VLSI can be classified into the following main categories, namely

a) Self-Checking design,

b) Online BIST,

c) Signature Monitoring in FSMs

d) Analog Methodologies.

Many of these schemes generate “Totally Self Checking” designs. However, several design and synthesis constraints are required by these methodologies to limit the scope of fault propagation that renders these methodologies difficult to be applied for high-speed DSM chips.

- 'Stuck-at' Faults

The fault model on which all test equipment and test vector generation strategies are based is the 'stuck-at' fault. This is a gross simplification of what can go wrong with a CMOS circuit, but suffices in practice. There are two possible 'stuck-at' states:
stuck-at-0, where a net remains low even when it is being driven high, and
stuck-at-1, where a net remains high even when it is being driven low.

Most fault models postulate that only one such fault can occur in a circuit at any time. This
implies that, for a circuit with \( n \) nets, the total number of different circuits, each with a single
'stuck-at' fault, is \( 2^n \). Any set of test vectors will identify some (between 0 and \( 2^n \)) of the
faulty circuits. The fault coverage of a set of test vectors is defined in terms of this number:

\[
\text{Fault coverage} = \frac{\text{Number of circuits with single 'stuck-at' fault identified}}{2^n}
\]

The ideal is to develop a set of structural test vectors which gives
100% fault coverage. This can be achieved in theory, on a number of categories of ideal
circuits, but is difficult in practice, and verifying it takes an unacceptable [19]

### 2.3.6 Design for Test

- **Ad-hoc Test Technique**

Ad-hoc DFT techniques are centered around identifying obscure nets in a circuit (ones
which are difficult to observe and/or to control) and inserting additional circuitry to link them
directly to primary inputs and outputs. Testability analysers can be used to identify such nets,
but if these are not available, a small number of rules of thumb can be invoked. The two most
useful are:

1. Break counter chains with test points
2. De-gate feedback loops

Counter chains are common in digital circuits. In many cases, the first counter is a
frequency divider, and the second a state counter. The problem with testing a circuit of this
nature is that the second counter only increments once for every complete cycle of the first. If
the frequency divider is an 8-bit counter, and the state counter a 4-bit counter, the complete
counting cycle is only achieved in \( 2^{(8+4)} = 4096 \) clock cycles.

![Counter Chain](image)

**Figure 2.8 Counter Chain**
If, however, a test point is inserted between the counters (Figure 2.9), with a link from the first counter terminal state to a test output, and a test input multiplexed into the second, then, in test mode with the test input high, both counters operate at the same frequency. The test cycle becomes that of the longer counter \(2^8=256\) clock cycles), and there are improvements to the controllability of the second counter and the observability of the first counter.

![Figure 2.9 Test Point in Counter Chain](image)

Ad-hoc strategies can be extended to take the full counter output to a set of test pins, and extend the counter to incorporate parallel loading, so that it can be initialised into any state. However, there is a significant overhead in test pins if these techniques are employed. The test pin overhead can be reduced by multiplexing test outputs with operational outputs, or by using bi-directional pads, which are inputs in operational mode and outputs in test mode (and vice versa). The minimum requirement is one extra pin, the test control pin.

- **Scan Path Design**

  A formal approach to design for test is to regard a circuit as a set of edge-sensitive storage elements connected by combinational logic blocks (Figure 2.10) A modified version of the storage elements, which can act as serial shift registers, is used in the circuit. These are connected to form a scan path, which is a single serial shift register threading the entire circuit.

  In test mode, data is shifted from the scan input through each storage element, to the scan output. The scan path test has the following cycle, which is repeated, a large number of times.

  Scan path testing has the advantage of treating each combinational block separately, using test vectors which may be specifically designed for it. Feedback loops are automatically

34
de-gated, counter chains are broken, and the logical depth of the circuit is greatly reduced. All storage elements are directly connected to a primary input and a primary output through the scan path. If the circuit contains large numbers of storage elements, the process of shifting data into and out of the scan path (flushing) can be slow, but it can be speeded up by having more than one scan path. The circuit can be reset in a single clock cycle by placing it in operational mode and using the global synchronous reset. Some CAD systems are able to build a scan path automatically, choosing the shortest physical route.

Figure 2.10 Circuit as Storage and Combinational Logic Blocks

A major advantage of scan paths is that it is possible to generate test vectors automatically, either with a pseudo-random sequence generator, or by an automatic test pattern generation program. This is because each combinational block of the circuit is tested separately.

The disadvantage of scan path testing is the silicon overhead in the serial/parallel latches, and the scan path routing. This can also cause some degradation of circuit function from the additional multiplexers in data paths. Scan paths cannot directly test level-sensitive elements such as RAM blocks.
The most time-consuming aspect of structural testing is the development of structural test vectors, and analytical or empirical tests to determine their fault coverage. An alternative built-in self-test approach is to incorporate on-chip circuitry which generates streams of random bits for use as test sequences. Statistical analysis has shown that these random sequences can achieve the same levels of fault coverage as sets derived analytically. The commonest source of an on-chip random bit sequence is the pseudo-random binary sequence (PRBS) generator, also known as a linear feedback shift register (LFSR). An example is shown in Figure 2.12. It consists of a shift register with its output gated with a tap point and fed back into its input. An appropriate choice of tap point (not a factor of the register length) gives, for n bits, a sequence of $2^n - 1$ bits. Different tap points produce different sequences.
Complementary to the PRBS generator is the signature analysis register (Figure 2.13), which accumulates an output test stream into a shift register, gating it with feedback from its own output and a tap point. This has the effect of compressing the output stream into a short signature, which, for an n-bit signature analysis register, will disclose a single-bit difference in a sequence of $2^n$ bits [19].

The combination of PRBS generator and signature analyser as a source and destination of test data on-chip gives ASIC built-in self-test (BIST) properties. The signature analyser can be linked via a comparator with a register containing the known fault-free signature, to provide a
single go/no go output. See Figure 2.13. An additional safeguard is to build in a way of introducing a single fault into the circuit, to test the go/no go circuit.

![Built-in Self-test Elements](image)

Figure 2.14: Built-in Self-test Elements

Built-in self-test techniques of this sort mean that there is no need to generate structural test vectors. The ASIC can be tested at any time, either in isolation or in-system. The cost is the additional circuitry, which is itself a source of potential fabrication faults.

- **Hybrid Techniques**

The DFT techniques described in the previous sections are useful in themselves for particular types of circuits, but their power is enhanced when they are used together. For example, scan path testing can be used with path sensitization and justification for each combinational block in a circuit to derive a set of test vectors with near-full fault coverage [19]. Testability analysis can be used to identify obscure nets for the insertion of test points.

One of the most powerful combinations is built-in self-test combined with a scan path. An example is shown in Figure 2.15. The PRBS generator feeds the scan path, the output of which goes to a signature analysis register. The signature can be multiplexed with operational data so that no additional output pads are required. A further integration of test and functional circuitry can be achieved by the replacement of storage elements with built-in logic block observer (Bilbo) registers. These can be in any of four modes: reset, scan path register, operational register or PRBS generator/signature analysis register. Although a Bilbo requires
far more gates than a register, it eliminates the need for a separate PRBS generator and signature analysis register.

![Diagram of Scan Path and Built-in Self-test](image)

Figure 2.15 Combinations of Scan Path and Built-in Self-test

This combination eliminates the need for test vector generation, has a high probability of coverage of most faults, and has only one additional pin, the test control pin. The test can be applied at any time using simple test equipment. The penalty is increased silicon area, a reduced peak operational performance, and the possibility of fabrication faults in the test circuitry. [19]

- **BIST: Built in Self Test (BIST)**

![Diagram of BIST Setup](image)

Figure 2.16 BIST Setup

When in self test, inputs to DUT is taken from LFSR and checksum register accumulates the output. After some X number of test vectors have been applied, compare check sum output against a golden value – if the same, then system has passed self test. Is Fault Coverage Sufficient Without Extra Hardware? Consider study results that show defect level vs. fault coverage
Figure 2.17: Defect Level Vs. Fault Coverage [15]

To get a single chip defect rate of 100 defective parts per million parts, you need 99.9% fault coverage! Does one really need a defect rate of less than 100 ppm?

A Solution: Scan Paths

• Is there a way to apply inputs directly to gates and observer outputs directly?

• Yes - with a Scan Path. A Scan Path ties all FFs in the design into a shift register.

requires scannable DFFs. Because of excitation and propagation of the crosstalk faults, in the detection of these faults, two successive test vectors are required and thus their generation is difficult. Moreover, it is necessary in the sequential circuit to consider the problem of timing such as gate delay time and FF setup time in the circuit. For that reason, it has been difficult to realize high fault coverage in the test vector generation methods proposed so far. To solve these problems, Built-In Self-Test (BIST) method which can detect the crosstalk faults. This method can be realized by a relatively small area overhead; moreover, by adopting a circuit configuration during simulation, which is different from the one during test, the simulation time is shortened. In this way, testing of large-scale circuits has been made possible. [15]

• Measuring Test Effectiveness

It has been the practice, for over three decades, to resort to the use of stuck-at models to imitate the effects of defects. This model was more realistic when SSI (small-scale integration) was predominant. However, the stuck-at model, for practical reasons, is still widely used by commercial tools. Basically, this model assumes that an input or output of a logic gate (e.g., an inverter, an AND gate, an OR gate, etc.) is stuck to a logic value 0 or 1 and is insensitive to signal changes from the signal that drives it. With this faulting mechanism the process, in rather general terms, proceeds as follows: Computer models of digital circuits are created, and faults are injected into the model. The fault-free circuit and the faulted circuit are simulated. If there is a difference in response at an observable I/O pin, the fault is classified as detected. After many faults are evaluated in this manner, fault
coverage is computed as Fault coverage = Number of faults detected / Number of faults modeled

Given a fault coverage number, there are two questions that occur: How accurate is it, and for a given fault coverage, how many defective chips are likely to become tester escapes? Accuracy of fault coverage will depend on the faults selected and the accuracy of the fault model relative to real defect mechanisms. Fault selection requires a statistically meaningful random sample, although it is often the practice to

2.3.7 Testing at manufacturing end

Things don’t always work as intended. Some devices are manufactured incorrectly; others break or wear out after extensive use. In order to determine if a device was manufactured correctly, or if it continues to function as intended, it must be tested. The test is an evaluation based on a set of requirements. Depending on the complexity of the product, the test may be a mere perusal of the product to determine whether it suits one’s personal whims, or it could be a long, exhaustive checkout of a complex system to ensure compliance with many performance and safety criteria. Emphasis may be on speed of performance, accuracy, or reliability.

The die, that pass the wafer test are packaged and then retested. Bonding a chip to a package has the potential to introduce additional defects into the process, and these must be identified. Diagnosis may be called for when there is a yield crash—that is, a sudden, significant drop in the number of devices that pass a test. To aid in investigating the causes, it may be necessary to create additional test vectors specifically for the purpose of isolating the source of the crash. For ICs it may be necessary to resort to an e-beam probe to identify the source. Production diagnostic tests are more likely to be created for a printed circuit board (PCB), since they are often repairable and generally represent a larger manufacturing cost. Tests for memory arrays are thorough and methodical, thus serving both as go–no–go tests and as diagnostic tests. These tests permit substitution of spare rows or columns in order to
repair the memory array, thereby significantly improving the yield. [11]

Figure 2.18: Flow of a typical ASIC through manufacturing test.

2.3.8 Yield Analysis for Digital Circuits / ASIC chips

Let us now look at yield analysis, based on various probability distribution functions. But, first, just how important are yield equations? James Cunningham [18] describes a situation in which a company was invited to submit a bid to manufacture a large CMOS custom logic chip. The chip had already been designed at another company and was to have a die area of 2.3 cm$^2$. The company had experience making CMOS parts, but never one this large. Hence, they were uncertain as to how to estimate yield for a chip of this size.

When they extrapolated from existing data, using a computer-generated best-fit model, they obtained a yield estimate $Y = 1.4\%$. Using a Poisson model with $D_0 = 2.1$, where $D_0$ is the average number of defects per unit area $A$, they obtained an estimate $Y = 0.8\%$. They then calculated the yield using Seeds’ model, which gave $Y = 17\%$. That was followed by
Murphy's model. It gave \( Y = 4\% \). They decided to average Seeds' model and Murphy's model and submit a bid based on 11\% die sort yield. A year later they were producing chips with a yield of 6\%, even though \( D_0 \) had fallen from 2.1 to 1.9 defects/cm\(^2\). The company had started to evaluate the negative binomial yield model \( Y = (1 + D_0 A/\alpha)^a \). A value of \( \alpha = 3 \) produced a good fit for their yield data. Unfortunately, the company could not sustain losses on the product and dropped it from production, leaving the customer without a supply of parts.

Probability distribution functions are used to estimate the probability of an event occurring. The binomial probability distribution is a discrete distribution, which is expressed as

\[
P(k) = \frac{n!}{k!(n-k)!} P^k (1 - P)^{n-k}
\]

If \( P \) is the probability of a defect on a die, then \( P(k) \) is the probability of \( k \) defects on the die, when there are a total of \( n = D_0 A_w \) defects, where \( A_w \) is the area of the wafer. The probability \( P \) is \( D_0 A / D_0 A_w = A / A_w \). Substituting into Eq. (1) yields

\[
P(k) = \frac{n!}{k!(n-k)!} \left( \frac{A}{A_w} \right)^k \left( 1 - \frac{A}{A_w} \right)^{(n-k)}
\]

To derive the equation for a die with no defects, set \( k = 0 \). This yields

\[
P(k = 0) = \left[ 1 - \frac{A}{A_w} \right]^{D_0 A_w}
\]

The first distribution that was frequently used to estimate yields was the Poisson distribution which is expressed as

\[
P(k) = \frac{e^{-\lambda} \lambda^k}{k!} \quad \text{for } k = 0, 1, 2, ...
\]

Where \( \lambda \) is the average number of defects per die. For die with no defects \( (k = 0) \), the equation becomes \( P(0) = e^0 \) if \( \lambda = 0.5 \), the yield is predicted to be 0.607. In general, the Poisson distribution requires that defects be uniformly and randomly distributed. Hence, it tends to be pessimistic for larger die sizes. Considering again the binomial distribution, if the number of trials, \( n \) is large and the probability \( p \) of occurrence of an event is close to zero, then the binomial distribution is closely approximated by the Poisson distribution with \( \lambda = np \).

Another distribution commonly used to estimate yield is the normal distribution, also known as the Gaussian distribution. It is the familiar bell-shaped curve and is expressed as
The variable $\mu$ represents the mean, $\sigma$ represents the standard deviation, and $\sigma^2$ represents the variance. If $n$ is large and if neither $p$ or $q$ is too close to zero, the binomial distribution can be closely approximated by a normal distribution. This can be expressed as

$$\lim P\left( a \leq \frac{x - np}{\sqrt{npq}} \leq b \right) = \frac{1}{\sqrt{2\pi}} \int_a^b e^{-u^2/2} du \quad \text{where } n \to \infty \quad \text{---------2.21}$$

where $np$ represents the mean for the binomial distribution, $\sqrt{npq}$ is the standard deviation, $npq$ is the variance, and $x$ is the number of successful trials.

When Murphy investigated the yield problem in 1964, he observed that defect and particle densities vary widely among chips, wafers, and runs. Under these circumstances, the Poisson model is likely to underestimate yield, so he chose to use the normalized probability distribution function. To derive a yield equation, Murphy multiplied the probability distribution function with the probability $P$ that the device was good, for a given defect density $D$, and then summed that over all values of $D$, that is,

$$Y = \int_0^\infty pf(D)dD \quad \text{---------2.22}$$

He substituted $p = e^{-D_0}$ for the probability that the device was good. However, he could not integrate the bell-shaped curve, so he approximated it with a triangle function.

This gave

$$Y = \left( \frac{1 - e^{D_0A}}{D_0A} \right)^2 \quad \text{---------2.23}$$

By substituting other expressions for $f(D)$ in Eq. (7), other yield equations result. Seeds used an exponential distribution function $f(D) = e^{-D/D_0}/D_0$. Substituting this into Eq. (7), he obtained

$$Y = \left( \frac{1}{1 + D_0A} \right) \quad \text{---------2.24}$$
In 1973 Charles Stapper derived a yield equation that is often referred to as a negative binomial distribution. By substituting \( p(x) = e^{-\lambda} \lambda^x / x! \) and the gamma distribution function

\[
f(\lambda) = \frac{1}{\Gamma(\alpha) \beta^\alpha} \lambda^{\alpha-1} e^{-\lambda / \beta}\]

into Murphy's equation [Eq. 7] and integrating, he obtained

\[
Y = (1 + D_o A / \alpha)^{-\alpha} \quad \text{------------------------2.25}
\]

The mean of the gamma function is given by \( \mu = \alpha / \lambda \), whereas the variance is given by \( \alpha / \lambda^2 \). Compare these with the mean and variance of the negative binomial distribution, sometimes referred to as Pascal's distribution: mean = \( nq/p \) and variance = \( nq/p^2 \).

The parameter \( \alpha \) in Eq. 2.25 is referred to as the cluster parameter. By selecting appropriate values of \( \alpha \) the other yield equations can be approximated by 2.25. The value of \( \alpha \) can be determined through statistical analysis of defect distribution data, permitting an accurate yield model to be obtained. [11]

### 2.4 Achieving Reliability through Testing

Growing complexity of digital systems has made reliability a major concern issue to designers. Testing adds cost to a product, but failure to test also adds cost. Trade-offs must be carefully examined in order to determine the right amount of testing [11].

#### 2.4.1 Reliability Engineering Basics

Reliability of a system is nothing but the probability that the given system will perform its intended function under specified conditions for a specified period of time. This is the probabilistic event and the engineers and designers are just giving the probable figures.

As reliability is dependent upon the conditions under which it is operating and the time of operation, so a more useful term is 'Mean Time Between Failures' (MTBF); this time is usually expressed in hours and is given by \( \int_0^\infty R(t) \, dt \), the area underneath the reliability curve \( R(t) \) plotted versus \( t \); this result is true for any failure distribution. MTBF is simply the reciprocal of the failure rate.

#### 2.4.2 Methods to Improve Reliability:

Fundamental approaches that can be taken to improve reliability are fault prevention and fault tolerance. The goal of the fault prevention system is to reduce the probability of system failures to an acceptably low value. In fault tolerant system, faults are expected to
occur, but incorporating redundancy automatically counteracts their effects. Fault tolerance is not a replacements but rather a supplement to the most important principles of reliable system design i.e.

1. Use the most reliable components and
2. Keep the system as simple as possible, consistent with achieving the design objectives. [10]

Building redundant subsystem is one of the main approaches used to improve the reliability of a System, though redundancy is not a desirable system design feature, it is one of the preferred approaches. Two limiting cases are frequently met in practice:

First case is a System in which each subsystem must be functional, if the system as a whole is to function. Another case is a System in which correct operation of just one subsystem is sufficient, for the system to function satisfactorily. In other words the system consists of redundant subsystems and will fail only if all subsystem fails. For commercial systems, a non-redundant, a fault prevention technique is preferred mainly because a redundant design results in extra costs. The reliability is improved by using reliable components, refined interconnections, but the approach is less viable in practice. A fault tolerant design can provide dramatic improvements in system availability and lead to substantial reduction in maintenance costs as a consequence of fewer system failures.

A Fault tolerant system is a system, which has the built-in capability (without external assistance) to preserve the continued correct execution of its input/output functions in the presence of certain set of operational faults. In the high-reliability applications tolerance of both anticipated and un-anticipated faults must be considered. Certain redundancies are introduced for the purposes of improving reliability are:

- Static Redundancy or masking redundancy
- Dynamic Redundancy – Cold-standby and Hot-standby
- Hybrid redundancy – Combination of Static and Dynamic Redundancy.
- Self – Purging Redundancy
2.4.3 Fault tolerant design for VLSI chips

Fault-tolerant designs of very large ICs primarily attempt to enhance yield. Such designs, first employed in memory chips, now encompass random logic VLSI and wafer-scale circuits.

The primary motive for introducing fault tolerance in VLSI circuits is yield enhancement, increasing the percentage of fault-free chips obtained. The active area of monolithic VLSI chips has always been limited by random fabrication defects, which appear impossible to eliminate in even the best manufacturing processes. Thus, the defect density in any fabrication line limits the size of the largest defect-free chip producible with commercially viable yields. Larger circuits demand a fault-tolerance capability to overcome fabrication defects while avoiding unreasonable costs. These two classes of defects contribute to yield losses. In mature, well-controlled fabrication lines, manufacturers can minimize gross area defects.

The yield loss due to random spot defects is typically much higher than the yield loss due to global defects. This proves especially true for large-area interconnects. Variety of fault-tolerant techniques with relatively small overhead have been proposed and successfully implemented in memory ICs. Most methods for incorporating fault-tolerance (that is, redundancy) into VLSI ICs have the following objectives in addition to their main goal of yield enhancement: [42]

1. No or very limited impact of the added redundancy on performance.
2. Equal or higher reliability.
3. Small additional area and power requirements.
4. Transparency to the user (after chip reconfiguration).
5. Fault-free ICs requiring no (or limited) additional manufacturing steps.
6. Defective redundant elements replaceable by other redundant elements.

The approaches followed to improve the yield by fault tolerant design uses one of the following approaches.

**Error-correcting codes:** Manufacturers frequently use error-correcting codes (ECCs) in large memory systems to mask intermittent faults. Thus, using ECCs for yield enhancement can contribute to reliability.

**Associative approach:** The spare row/ column approach applies only to the replacement of individual faulty rows or columns. If it needs to replace larger blocks of cells, as partially good chips.
Partially good chips: A different approach to yield enhancement suggests the use of partially good chips. If sections in a 1 megabit memory are defective beyond repair, we can reconfigure the chip to a usable 0.5-megabit chip or even a 0.25-megabit.

2.5 Testing issues in VLSI systems

The cost of VLSI ICs mainly depends on Engineering efforts to generate test set and to carry out the test at the time of manufacturing. A general thumb rule is that capital costs run in the range of 50% of the overall IC test cost in the industry, so looking at capital costs is an essential analysis for manufacturing test. Figure 2.19 shows a plot extrapolated from the 1997 SIA technology roadmap for semiconductors [1]. It shows the capital costs for chip fabrication versus the capital costs for manufacturing test, normalized per transistor. The top curve shows the consistent reduction in chip fabrication cost per transistor that is the basis for Moore’s law, which in turn drives the continued expansion and evolution of the semiconductor business. The bottom curve, which can be traced back 20 years, indicates capital expenses for IC test have been essentially flat per transistor. Based on the 1997 SIA data, this trend of flat test capital cost per transistor was projected to continue for the foreseeable future.

Figure 2.19 Moore’s Law for Test: Fab. vs. Test Capital
While the cost of silicon real estate in a high integration die or SOC is often nicely predicted by die size and defect density, the effort and tools needed to "cut and paste" different design blocks and types together is often underestimated. More often, the effort and cost for design validation and manufacturing test of re-used design blocks and high integration ICs is much greater than the sum of those for the original designs blocks.[40]

2.5.1 Field Programmable Gate Array (FPGA) Design Flow

A FPGA is an array of regular logic blocks, which may be configured to operate within a particular logical function. Digital logic functions can be synthesized onto the logic blocks of the FPGA and it is then programmed with the configuration information to perform that function. FPGA devices are economical at small volumes because the non-repetitive engineering (NRE) costs are lower than with an Application Specific Integrated Circuit (ASIC). FPGAs are reprogrammable with lesser tool costs for development and design flow as compared to ASICs.

FPGA part vendors have traditionally provided FPGA design software. However, advances in fabrication technology have seen an explosion in FPGA logic capacity, and this has moved the complexity beyond software provided by FPGA tool vendors. ASIC tool vendors have now moved into the market place to provide very high quality front-end tools to the FPGA market at a reduced cost.
Cross-talk, impedance mismatches and manufacturability are typical problems in FPGA board design. They are often discovered late in the design cycle when the design grows and clock frequencies increase.
2.5.2 Design flow - concurrent – HDL based approach

The Design Phases:

- **Functional Specification:** At the beginning, the designer has to specify the functionality of the system. Basic blocks of the hardware are identified and their interfaces, composed of data and control signals, are fixed.

- **Behavioral Implementation:** The goal is to develop a simulation model as quick as possible, to discover logical design errors in an early design phase. The model reflects only the logical behaviour, it doesn’t include any timing information (or only rough estimations). The formats mostly used today for designing hardware are **Hardware Description Languages** (HDL). Verilog and VHDL are the prevailing languages in this area.

- **Different Abstraction Levels:** The different abstraction levels offered by HDLs are:

  - **Behavioral:** Behavioral modeling (also referred to as functional or system-level)
RTL-level: Register-Transfer-Logic refers to designs, which use only the subset of an HDL suitable for synthesis.

Gate-level: HDLs offers build-in primitives for the standard logic gates like and, or, nor, xor, not, buf etc. Gate-level (structural) code is based on these primitives and their interconnection through wires.

- **Schematic Entry:** Usually, a design is structured hierarchically, composed of modules with functionality and modules interconnecting them. It can be specified in HDLs.

- **Logical Simulation:** To make sure that the design behaves as specified, some test patterns have to be generated and applied to the inputs of the model. Comparing the produced outputs with the ones expected, the correct logic behaviour of the model can be validated.

- **Synthesis:** In general, synthesis describes the transformation from one format into another. This design phase can be split into high-level synthesis and logic synthesis.

- **Gate-Level Simulation:** After a design is synthesized one has to verify that the model still meets the requirements of the design goals. Perhaps signal paths are created, where the logic transition of a signal arrives later at a register stage than the active clock edge.

- **Floor planning, Place & Route:** During floor planning, the logical design is partitioned into physical blocks to take advantage of the whole chip area efficiently. The next step is to fill these blocks with specific cells, which are placed with respect to the designers constraints. Finally the connections between the physical blocks are routed.

- **Fabrication:** At the end of the whole process the transistor-layout is put into silicon. According to the long duration of an IC process (8-12 weeks) and its high costs it is of great importance that all design errors are detected during the simulation phases. [44]

Verilog and VHDL have become the standards for expressing designs at all levels of abstraction, although investigation into specification languages continues to be an active area of research. Its importance is seen from such statements as “requirements errors typically comprise over 40% of all errors in a software project” and “the really serious mistakes occur in the first day.” A design expressed in an HDL, at a level of abstraction that describes intended behaviors, can be formally tested. At this level the design is a requirements document that states, in a simulation language, what actions the product must perform.

The HDL permits the designer to simulate behavioral expressions with input vectors. Design automation chosen to confirm correctness of the design or to expose design errors. The design verification vectors must be sufficient to confirm that the design satisfies the
behavior expressed in the product specification. Development of effective test stimuli at this state is highly iterative; a discrepancy between designer intent and simulation results often indicates the need for more stimuli to diagnose the underlying reason for the discrepancy.

A growing trend at this level is the use of formal verification techniques. The logic design is tested in a manner similar to the functional design. A major difference is that the circuit description is more detailed; hence thorough analysis requires that simulations be more exhaustive. At the logic level, timing is of greater concern, and stimuli that were effective at the register transfer level (RTL) may not be effective in ferreting out critical timing problems. On the other hand, stimuli that produced correct or expected response from the RTL circuit may, when simulated by a timing simulator, indicate incorrect response or may indicate marginal performance, or the simulator may simply indicate that it cannot predict the correct response.

The testing of physical structure is probably the most formal test level. The test engineer works from a detailed design document to create tests that determine if response of the fabricated device corresponds to response of the design. Studies of fault behavior of the selected circuit family or technology permit the creation of fault models. These fault models are then used to create specific test stimuli that attempt to distinguish between the correctly operating device and a device with the fault. [11]

### 2.5.3 Design flow - real time – Emulation

Design verification has a large impact on the final testability of a system. The identification and removal of design errors from the initial design steps increases the testing quality of the entire design flow. An emulator accelerates a validation methodology for RTL designs. Alternative emulator configurations are made available by researchers to evaluate the performance speed-up of the presented methodology. In fact, the target errors are not design specific (i.e. microprocessor or memory cell errors). In order to download the design under test (DUT) description into the emulator, it has to be synthesizable. This requirement has implied the definition of injection functions, guarantees the synthesizability of the faulty DUT. Fault injection functions are classified by the testing community as mutants. They are HDL fragments, which can alter the design behavior simulating design errors. Each emulated fault is activated by driving an additional input port added to the DUT during the fault injection process. Each fault is identified by a unique fault ID. Driving the added port to the
fault ID of the selected fault, the automatic test pattern generation procedure can try to generate test sequences for the faulty configuration. The defined emulation-based testing architecture is shown in Figure 2.22. A copy of both, faulty and the fault-free design are downloaded into the emulator. [45] . The ATPG engine generates the test sequences, which will be applied either vector by vector or sequence by sequence on the basis of the selected emulator running mode to the design primary inputs. Whenever a design error is detected, the emulated version of the DUT has to be corrected. The testing loop is iterated until no more design errors are detected. [45]

![Figure 2.22 Emulation based testing architecture](image)

2.6 VLSI Testing tools

- Test tool requirements

  Because of the wide range of test techniques that must be supported, the test automation tools must support a broad range of test types. The tools must keep track of the different test types and understand which manufacturing tester can handle each type of test. The test patterns for each type of test must be clearly identified and kept separate. The test coverage for each type of test must be accounted for and accumulated with coverage for other tests where appropriate. Much of the complexity can be hidden by using DFT synthesis tools, such as the DFT insertion products available from IBM EDA. This can quickly convert a nonscan design to full (or partial) scan, and can insert IEEE 1149.1 boundary scan [13] and/or IBM boundary-scan structures into a chip design. This can remove much of the
manual effort from the DFT for a chip and can shorten a design cycle by many days. Many studies have shown that high stuck-fault coverage does not necessarily imply high coverage of potential defects [14]. Thus, being able to target known defect mechanisms that do not manifest themselves as pin stuck at faults is essential to ensuring that the generated test vectors are the best possible.

2.6.1 Test benches and pattern generators

- Testbench Components

The testbench consists of a few components: the stimulus, the stimulus conditioning, the system under test, the load and the results extractor. The system under test is merely the system that is being analyzed by testbench simulation. The stimulus is needed to excite the system under test and elicit a response. The load is required to add an output capacitance to the system under test in order to create a test environment that more realistically parallels an actual system. The results extractor monitors the inputs and outputs of the system under test and determines what is relevant.

![Test bench Block Diagram](image)

- Stimuli

Stimuli can be created in several ways. In Matlab, a stimulus is merely a variable dependent on time, frequency or some other domain depending on the type of testbench is being run. Since all simulations are not in continuous time, but are discretized into samples, the stimulus in Mat lab is really just a vector and can be implemented as one. Verilog AMS stimuli are similar to Mat lab stimuli, however, they are always a function of time because transient analysis is the only analysis of which Verilog AMS is capable. Like Mat lab, Verilog AMS has several math functions that can be used to more easily implement the
stimuli. Unlike, Mat lab, Verilog AMS can not pass vectors to the math functions. Therefore, the signal must be constructed in time steps that are usually divided into picoseconds by the Verilog AMS compiler. Verilog AMS modules will be our primary means of stimulus creation.[19]

**Test-bench Reusability**

The term “design reuse” is prevalent in the fast-paced, time-to-market driven technology industry. Products must be designed, tested, and manufactured faster than ever to meet a technology-driven market economy bent on having the latest technology toys. Industry has realized that reusing building blocks of intellectual property, IP, test-benches from previous designs or developed during the process of designing at earlier stage is one way to get ahead on the project timeline.[20]

However, a quick survey from an audience at the ’99 IHDL Conference done by Cliff Cummings showed that only a handful (less than 10%) of engineers liked to write test benches and test code to verify the design. In general, many engineers re-engineer a design but not going out of their way to reuse the existing test bench, even if it is not efficient, well documented, or even up to specification!

![Effort Spent to Re-Engineer a Reusable Module](image)

Figure 2.24: Engineering work required when reusing Design / Testbench

- **Poor Forms of Test Benches:**

  One can “shoot himself in the foot” multiple times when implementing test benches. If a well thought, high-level testing strategy is not followed, the test bench reusability gets
hindered to the point of non-reusability. Some examples of poor test benches for re-use would include any of the following, ordered from worst to workable-but-ugly methods. [20]

1. **Vector Stimulus:** The ability to understand what the vectors are doing (i.e. documentation of the stimulus) as well as the ability to modify the tests to incorporate bug fixes or design improvements is lost due to the low level format. The worst possible use of vectors is to create a “golden set” of test vectors.

2. **Assembly Language Code:** Assembly code generally means a lower level of abstraction of the test and limits the engineer in easily creating a functional test description to perform large, complex testing operations. The assembly language code test bench will work, but the effort to re-use it requires more overhead in terms of tools used to compile the assembly to object code and the effort to create the test. By using an assembly-code driven test bench, reusability gets limited to a platform-specific tool for code compiling.

**Scripts and Environments:** EDA tools are never perfect, and no testing solution will always fit the requirements. To patch problems at hand, the engineer winds up creating a script-based workaround, usually in perl. What can turn a test bench into a non-reusable nightmare is the when engineers break away from an industry standard, widely used, HDL language (VHDL, verilog, C/C++) to do the testing and create a whole environment of support scripts, test language scripts, and pre/post-processing scripts.

- **Making a Reusable Test Bench:**

A reusable test bench requires an “Open Test Bench” philosophy. Ideally, this means that a minimal set of tools that are industry-proven, and widely available using standard, industry known design languages utilized to implement and run the test bench. [60]

1. **Test Bench in Widely-Used RTL:** For maximum reuse of tests, the test language should be ideally in the HDL (VHDL or verilog) of the chosen RTL simulator. The design is using a HDL to describe the RTL/behavioral code at a high-level of abstraction, why shouldn’t the test bench be similar? The engineer reusing the design will generally have the same simulator as the code is written in and a good understanding of the HDL in use. The test is more portable to other same-HDL simulators, easier to understand, and faster in simulation because it is usually compiled into the simulator as a native language.
2. **High Level Testing Methods**: By writing at a higher level of abstraction, the test code is very portable and can be applied toward testing any abstraction of the design, such as RTL and gate-level, back-annotated-from layout timing netlist.

3. **The use of C/C++**: The one exception to the "same test language as RTL rule" is the use of C/C++ through PLIs. As most engineers are painfully aware of, verilog is limited in the text I/O area. Using PLIs to provide useful I/O and string facilities like scanf, sprintf, etc., greatly enhance the test benching ability. Testing SOC designs where large efforts are placed in software-based testing can be accelerated by the use of software-created C/C++ test code through the PLI interface.

### 2.6.2 Result generation methods

- **Results Extractor**

  The results extractor monitors the inputs and outputs of the system under test to calculate and store relevant simulation results such as system transit time, system gain, frequency response and linearity. Generally, a results extractor module will be created along with a stimulus module and may not be compatible with other stimuli.

- **Noise**

  Modeling noise is important for doing bit-error rate and noise figure analysis for analog/mixed-signal systems. Noise is typically modeled as additive white Gaussian noise (AWGN). Therefore, it is important for Verilog AMS to have a random number generator that can generate numbers in a Gaussian distribution. AWGN has a flat and infinite bandwidth along with a Gaussian distribution.

  The new system, called TestBench™ (IBM product) is one of the tools which is basis for the test structure verification, test-pattern generation, and fault grading functions. TestBench™ has been commercially available since early 1994. TestBench™ currently supports the following types of test data. [18] Other commercially available testbenches are supporting the same features more or less.

  **Stored-pattern**

  Static or dynamic logic tests:

  - Static or dynamic embedded macro tests.
  - Shift register flush and scan tests.

  **Parametric tests.**
I/O wrap tests.

Interconnect tests (for multichip modules or boards).

Static or dynamic Weighted Random Pattern (WRP) logic tests.

Static or dynamic logic BIST.

Static or dynamic embedded macro BIST.

Signature-based

2.6.3 Discussion on tools

Variety of tools are needed in VLSI design for simulation, synthesis and verification.

All synthesis tools tend to be constraints-driven (timing, area, power) to reduce number of possible iterations. Still some work is done manually. Verification tools are indispensable at all abstraction levels for correctness. Still more efficient approaches sought.

Tools are available from different vendors which aids to the design process. Every tool is having few unique features and of course, few limitations also. Varieties of tools are available for Simulation, Synthesis and back-end design.

Xilinx is one of the major players in the area, the company provides tools to accomplish the entire design flow based on VHDL. VMS is another key player in the area, with verilog and Set of Tools. Mentor Graphics is providing certain good simulation and synthesis tools.

Cadence is providing certain tools which includes “IC-Tools” => IC5141 package (Linux), Signal Storm => TSI42 package (Linux), Abstract Generator => DSMSE54 (Solaris), First Encounter => SOC42 package (Linux)

Collection of tools managed by Design Framework II (DF-II) includes Virtuoso schematic/layout editor, Analog Environment, Spectrum simulator, Diva DRC, EXT, LVS

Synopsys is providing Design Compiler (Linux), Power compiler

• Mentor is providing HDL Designer (Linux)
2.7 Needs of ASIC testing

2.7.1 Pre-processing before putting to FAB

**Boundary Scan** is a structural test method, whereas CPU emulation is a functional test method. Functional tests, although sometimes difficult to generate, will cover some (usually unknown) structural defects whereas structural tests are unlikely to cover any functional misbehaviors. Discussions between ASSET InterTech and International Test technologies showed a potential synergy between the boundary scan and CPU emulation approaches. That is, use boundary scan to structurally test a board, and then use CPU emulation to structurally test those parts not covered by boundary scan, and to implement a basic functional test of the board.

- Example Application

To illustrate the potential synergy between classical structural board test based on boundary scan and emulation functional board test based also on boundary scan using eJTAG ports, we will explore the possibilities on the processor-based board shown in Figure 2.24

![Figure 2.25: Processor based board](image)

In Fig. 2.25, the chips with 1149.1 boundary scan are shown. The board areas covered by boundary scan test are primarily between the boundary-scan devices. Tests are structural only and are generated from CAD data. Diagnostics are to node level. Structural defects associated with the SDRAM and Flash devices can also be tested. Fig. 2.25 shows the test coverage of CPU emulation. Tests are primarily functional but there will be indirect coverage.
of manufacturing defects. Tests are generated manually or semi-automatically, and diagnostics are to functional level.

![Diagram of a processor based board with boundary scan and non-scan areas](image)

**Figure 2.26** Processor based board with boundary scan and non-scan areas

![Diagram of a processor based board with CPU emulation](image)

**Figure 2.27:** Processor based board with CPU emulation

Although full test coverage is available via CPU emulation, boundary scan can do a better job on certain functions when it is available. This is because test development can be driven from CAD, interconnect pattern generation is fully automatic, and diagnostics usually identify the exact location of opens and shorts. With CPU emulation, test development is a semi-automated process, and diagnostics only identify faulty functions, not nodes. A combined approach, which optimizes test coverage, test development time, and diagnostic resolution is a preferred approach over both the methods [16]
2.7.2 Testability Provisions for Post-FAB Testing

The methodology is based on level-sensitive scan design (LSSD) plus boundary scan and special test controls; software aids are used to insert LSSD and other DFT features into the design, to check the design, and to generate and fault-grade test patterns. This test methodology, combined into a user package for an entire sign-off process, makes the tools, technology, and design flow easier for the designers to use and understand.

- **Reduced-pin-count testing**

  It is based on using LSSD boundary scan to permit the use of relatively low-cost testers containing fewer full-function pin channels (ac pins) than the number of signal I/O (input/output) pins on the ASIC devices being tested. This allows using older 256-pin testers and with internally developed 64-pin tester to test ASICs with a variety of signal counts, up to 2000. This method is also referred as “reduced-pin-count testing”; it can save substantially on tester costs, since the cost of logic testers is nearly directly proportional to the number of tester pins, and ever-increasing signal counts can lead to underutilization of existing investments in older testers.

- **Random-pattern tests**

  To improve quality and reduce test data volume, pseudorandom pattern generators (PRPGs) are incorporated into tester hardware to produce a variable distribution of logical 1s and 0s for each test-pattern input bit. This method selectively biases, the test-pattern inputs to a greater probability of 0 or 1. High test coverage can be achieved, since random-pattern-resistant faults can be tested in a reasonable number of patterns. Furthermore, RP testing also involves compressing the outputs into signature registers. One potential drawback to RP testing is that it can result in far more patterns being applied than with stored pattern tests. This may be good for detecting unmodeled defects.

- **Input and output tests**

  Chip buffers must be tested to ensure that they operate correctly for their specified voltage range. These tests are called driver tests; they ensure that the input, output, and three state buffer meets its voltage specifications.
• **Stored-pattern ATPG tests**

  Chip internal testing can be done with stored patterns. Stored patterns can be applied on any logic tester, thus allowing some testing to use older existing testers. Stored patterns can be used to test chips that do not conform to the boundary-scan signature-based testing requirements. Also, stored patterns are used for the driver and receiver tests. Stored-pattern tests can be either static or delay tests, though they are mostly used as static tests. By the use of full-scan in the chip designs, ATPG can achieve a very high (99.5%+) stuck-fault test coverage.

• **Transient Signal Analysis (TSA) (\(I_{dd}\) based test method)**

  It is a parametric testing method for digital integrated circuits. In TSA, transients in both the voltage waveforms at selected test points as well as current transients on the power supply are analyzed to determine the presence of defects. TSA exploits the fact that the power supply is globally connected to a large fraction of the transistors in a CMOS digital integrated circuit. TSA is similar to power supply current (IDD) test methods in this way. Since power supply connections are un-buffered at the I/O pads, it is possible to measure the high frequency components of this signal without attenuation. Thus, the IDD transients reflect the switching activity associated with the propagation of signals throughout the circuit. TSA improves on defect detection capabilities of IDD test strategies by additionally monitoring the voltage transients at a set of test points as well as on IDD. By using a combination of voltage transients on test signal paths and IDD transients on the power supply, TSA can provide improved defect resolution while maintaining a high degree of process insensitivity when compared with other IDD-based testing methods. [18]

• **Burn-in**

  Because some types of defects cause semiconductor chips to fail early in their expected life, To weed out these defects, the high-reliability chips are generally “burned in” (patterns are run through the chip at an elevated voltage and temperature). This simulates the early life of the chip and brings out defects that can shorten chip life. By applying test patterns that achieve high stuck-fault coverage in burn-in conditions, soft defects that occur only at higher temperatures can also be detected. [17]

• **Production Readiness Review**

  Pre-Processor ASIC (PPrASICs) are delivered as wafers from the foundry. Given the parameters of the production-process and the physical size of a single die, the factory’s
estimate for the yield of "Known Good Dice" (KGD) is approximately 60% of all dice produced. It is obvious, that ASICs cannot be mounted blindly onto Multi Chip Modules (MCMs), because the fraction of "throw-away" MCMs equipped with additional components would be huge. Such proceedings are forbidden, not only from the financial point of view. Hence, it is mandatory to perform tests on the wafer-level, which strongly enhance the supply of KGDs for MCM-mounting. Another task, just as important, is the full functional testing of the finished product. Essential for testing is the VME-infrastructure, which provides computer-controlled input generation and output capture.

The hardware was specially developed as test-bed for many purposes. The effort made to have a versatile test-environment pays off in a sense: the VME configuration as seen in Figure 2.28 including the CMC daughter-boards, is a well-adapted tool to perform the tests

![Test Set-Up](image)

Figure: 2.28: The Hardware Test Environment

A menu-driven software-package is written to perform the test-tasks:
1) To test “naked” dice as delivered from the foundry in a wafer.
2) To test “finished” MCMs as delivered from the company after mounting.
• Marking “Known Good Dice” for selection after cutting.

The left map of the wafer shows results from a “first go”. It shall be noted here, that testing also reveals problems, which are not related to the item under test, but to the testing tools themselves. While performing the test, it was discovered that the “chuck” (wafer holding test-table of the probing station) has a slight (few ten microns) inclination. This leads to reduced needle-pressure. The effect can be spotted, when looking at test-results on the left side of the main diagonal across the wafer.

Correction of inclination and readjustment of needle-pressure leads to results as seen on the map to the right. Such wafer maps will be used to mark “bad dice” by “dotting” them with an appropriate tool (automatic “inker”).

![Wafer-maps showing test-results as colour-code](image)

2.7.3 Post- FAB testing of ASIC

Post-fabrication testing is carried out to check that each manufactured die does not contain any faults. These tests are developed on a simulation of the ASIC, during design development. Simulation is one of the facilities offered by most CAD suites. It can be done immediately after design entry, in which case each part of the circuit is simulated in isolation. In most CAD systems, a loaded simulation can be carried out after physical layout, taking into account interconnect loadings. This gives a good idea of the operational behaviour of the fabricated ASIC.

Simulations are driven by sets of test vectors, which are applied in sequence to the circuit inputs, while the state of the outputs and (optionally) of certain internal nets (wires in the circuit) is monitored. Test vectors for functional testing are fairly straightforward to
develop. They consist of typical input patterns that the ASIC would experience under normal operational conditions, plus a few extreme cases to test it with limiting data values.

Test vectors to identify fabrication faults (also known as structural test vectors, as they are based on the structural properties of the ASIC) are much more difficult to develop. They are nevertheless an essential element of the ASIC design cycle. Inadequate structural test vectors can mean that faults in the fabricated die go undetected, with potentially serious consequences. It is important to consider the testability of the ASIC from its inception, and incorporate a design for test (DFT) strategy into the design.

However, generating post-fabrication test vectors follows a law of diminishing returns. To generate a set of test vectors which gives full fault coverage (if it is theoretically possible) can require several times the design time required to generate a set which is regarded as adequate for the task. The strategy for test vector development needs to take a number of costs factors into account, including those of design, computer resources, fabrication, wafer test, packaging, component test, board test, system test and field test. All of these need to be balanced in relation to the size of the production run [1].

A testing strategy needs to be incorporated into the overall design cycle. The ASIC supplier performs the post-fabrication tests using the functional and structural test vectors developed by the designer. Parts which pass these tests are released to (and paid for by) the designer. The fuller is the post-fabrication test, the fewer parts that are subsequently rejected by the designer after in-system tests.