Definition of a description language for IDDQ monitors

Waayers, T.F.

Award date:
1995

Definition of a description language for I_{DDQ} monitors

by

T.F. Waayers

June 1995

supervisor:	Prof. ir. M.T.M. Segers
mentor:	M. Sc. K. Baker
advisor:	Ir. F.G.M. Bouwman

Definition of a description language for I_{DDQ} monitors

Abstract

This report describes the development of a language based on VHDL intended to simplify the use of I_{DDQ} instrumentation in production testing. This language called 'Monitor Description Format' or MDF, is part of the development by QTAG (Quality Test Action Group) of an infrastructure for quiescent current testing. The MDF can be used with commercial supported QTAG monitors (I_{DDQ} measurement equipment) to create a 'Plug & Play' environment for I_{DDQ} testing. Prior to the definition of MDF, first all known available I_{DDQ} monitors and QTAG proposals are discussed since developments in monitor design have a great deal of impact on the semantics and syntax of MDF. Furthermore, MDF integration in the Philips Test-development flow is discussed. This integration reveals some interesting shortcomings of todays Test-development tools with respect to I_{DDQ} testing.

Keywords

CMOS testing , quiescent current , I_{DDQ} monitoring , Automatic Test Equipment , QTAG

Preface

This work was performed in partial fulfilment of the requirements to become Master of Electrical Engineering at the Eindhoven University of Technology. It was performed in the group 'VLSI Design Automation and Test' at Philips Research Laboratories during the period from October 1994 up to June 1995.

Acknowledgements

I would like to thank a number of people in supporting me in this project. My mentor Keith Baker for his excellent guidance and advice. Frank Bouwman for his advice on software development, his punctual report reviewing and his overall support. Math Verstraelen for his support on testing in general. Furthermore, I would like to thank my supervisor Prof. ir. Rene Segers and Eric van Utteren for giving me the opportunity to carry out this project at Philips Research. I would also like to thank my roommates and fellow students who made my stay at the Philips Research Laboratories very pleasant.

Note

Section 2.2 of this report has been taken from [Bouwman, 95].
Section 6.8 of this report has been taken from [Baker, 95].

Philips Research Laboratories,

Eindhoven, June 1995

Contents

1 Introduction

Testing of Integrated Circuits (ICs) has become a major issue in research and development of Very Large Scale Integration (VLSI). ICs have been growing continuously in number of transistors and complexity. This has caused an increase in the probability of design errors. To prevent design errors, the design must be checked at several stages of the design. It is not possible to prevent manufacturing faults. Because the market asks for reliable, zero defect ICs every single IC produced must pass several tests. One of these tests is the *structural test.* The idea of structural test is that all structures, created on the silicon surface, must be tested for correctness against known models for failing the process. The problem of structural testing of ICs is complicated by the enormous amount of possible faults, and the limited accessibility of parts of the ICs via the IC pins.

To detect as much faults as possible during structural test, it is generally believed that already during the design phase of the IC, testability has to be taken into account. This is referred to as Design for Testability (DfT). In DfT, methods are developed which give greater *controllability* and *observability* to VLSI circuits under test. In case of combinatorial logic, this can be achieved by automatic test pattern generation tools, using algorithms based upon fault models. The complexity of getting high controllability and observability within sequential circuits demands for extra precautions during the design trajectory of an IC. The sequential controllability and observability is achieved by memory elements which can be used in two modes, called a scannable memory element. In normal mode, it acts as a conventional memory element. In test mode, the input of the memory element is directly under control of the vector generation by test software. All scannable memory elements in test mode form a shift register, called a *scan path*, which can be filled with stimuli data. After a period in normal mode, in which the device acts according its specification, the contents of all scanable memory elements can be shifted out in test mode.

When the scan test DfT technique is used during the design phase the testability of the design improves drastically, resulting in high fault coverage within relative short test times. This is very important because structural test should be performed on every individual IC produced.

While testing, stimuli are applied to the IC bringing it into a predefined state. After a 'normal' period, resulting values are captured and compared with known responses. ICs failing this comparison intend to be defect. The generation of test stimuli and related expected data is done according to fault models, for example the stuck-at model in which every fault on the input or output of a gate is assumed to behave as if it is shorted to fixed logic values. In the case of CMOS, many defects do not cause stuck-at behaviour. Other models like 'bridging and open fault model' are needed to generate tests with high fault coverage.

Conventional testing is done by so called *voltage testing*. During this type of testing responses of the device under test (DUT) are captured and compared with expected data, as mentioned above. To achieve higher fault coverage with CMOS VLSI ICs, another test method called *current testing* is used. This method is based on the fact that CMOS ICs cause very low power supply current during the logic quiescent period. This current, I_{DDQ}, can be nanoam-

peres for VLSI circuits and much less for smaller scale ICs. In CMOS ICs, stuck-at faults and most other types of defects cause state-dependent elevated I_{DDQ}. Consequently, I_{DDQ} measurement is a very efficient test technique for CMOS ICs. To achieve highly reliable ICs, I_{DDQ} should be measured at each test vector in a test set that achieves 100% node toggling [Rajsuman, 95]. Despite its effectiveness, CMOS IC test methods at most companies usually measure I_{DDQ} at only a few test vectors or not at all. This is partially due to the lack of commercial instrumentation for high-speed I_{DDQ} measurements (current monitors) and partially due to the difficult technique: "Hearing a pin fall after a cannon shot !".

At the present time, development of a current monitor is an open problem. A number of companies are developing current monitors independently to match with their existing test equipment and production lines. Considering the present difficulties in current-monitoring systems and the mismatch in various equipment, the IEEE technical committee on test technology has initiated the development of a standard for current monitors. The proposed name for this standard is QTAG (Quality Test Action Group). One issue of this standard is the so called 'Monitor Description Format' (MDF), a description language that uses the syntax of VHDL, IEEE Std 1076, in the way BSDL (Boundary Scan Description Language) does [Parker, 90] to describe the functionality of the I_{DDQ} monitoring hardware. MDF is needed to allow the monitor to be described in a machine and human readable format, so that different monitors could be used with different test systems, and would allow the test engineer to efficiently and effectively utilize the monitor in the application.

This report deals with the definition of the MDF and the use of MDF in the I_{DDQ} test development flow to create a 'Plug&Play' environment for I_{DDQ} monitors. To show the possibilities of MDF, a compiler has been written using LEX and YACC. This compiler has been integrated in the Philips test-development flow by the developed MDF2TASS software, using the possibilities of the Philips test assembler software TASS [CAT team, 94].

In the first three chapters of this report an introduction to testing and testing hardware is given for those who are not familiar with testing. In the next chapter QTAG and its proposals are introduced. After these introductory chapters a start for the development of the 'Plug&Play' environment is made in chapter 6 by mapping the currently available monitors. In chapter 7 the definition of MDF is described. The following chapters describe the I_{DDQ} test-development flow and the development of the MDF2TASS software used in the Philips test-development flow. Also results of an example are given. In the last chapters conclusions are drawn from the MDF2TASS results and subjects for further research are mentioned.

2 IC Design and Testing

2.1 Introduction

Before an IC can be delivered to a customer, its correctness must be determined. This implies that each individual IC must operate conform to its specifications. Since the design of ICs is done in a number of phases, it must be checked that no errors are introduced (an error meaning a difference between implementation and specification). After manufacturing the IC must be tested to assure its functionality conform its specification. The term *defect* is used to mean those physical occurrences which give rise to some circuit malfunction. A *fault* is the logical manifestation of a defect. In this chapter we will first discuss the IC design process in more detail. Secondly, we will take a look at the IC testing process, which is closely related to the design process and *verification* as opposed to *production testing* will be explained.

2.2 The IC development flow

The implementation of a concept at system level into one or more ICs consists of many small steps [Claassen, 89][Bruls, 94]. One way of representing this development is illustrated in Figure 2-1. The figure shows the flow from high-level to low-level specifications, the manufacturing, and the various levels of testing.

2.2.1 Design

Developments start with a concept for a system. Once the partitioning of the system in the various ICs is done, a requirements specification can be written for each IC. This is often not a formal description using a specific hardware description or programming language, but a plain textual description of the high-level behaviour of the complete IC.

The second step is to translate this requirements specification into a formal functional description. On this level, the functionality of the complete IC is described on a level of, for example, adders, multipliers, and registers. For this purpose, formal hardware description languages, such as VHDL, or other programming languages (Pascal, C) are used. The functional specification can be simulated to verify the correctness. However, the step between requirement and functional specification is a weak point in the flow and remains a source of concern because formal verification of this step is impossible due to a lacking formal requirements specification.

The formal description at the functional level can be used to generate the structural specification (gate-level description) and finally a layout. It is common practice to apply commercially available synthesis tools for these last translation steps. Verification of the correctness of these steps is possible as a result of the formal approach.

2.2.2 Manufacturing

The resulting layout specification has to be implemented on silicon. For this purpose each layer of the layout is handled separately and several processing steps are required for each of them. It is important to notice here that, due to the complexity and sensitivity to disturbances of various processing steps, it is possible for an IC to contain a defect causing it to fail one or more specifications [Maly, 88a]. Depending on the size of the chip and the stage of development of the IC as well as the process, the percentage of fault-free processed VLSI circuits, i.e. the production yield, can virtually have any value. For complex innovative ICs and immature processes it may be low, 10% or even less, while for mass-produced ICs in a mature process the production yield may be as high as 95% or even more.

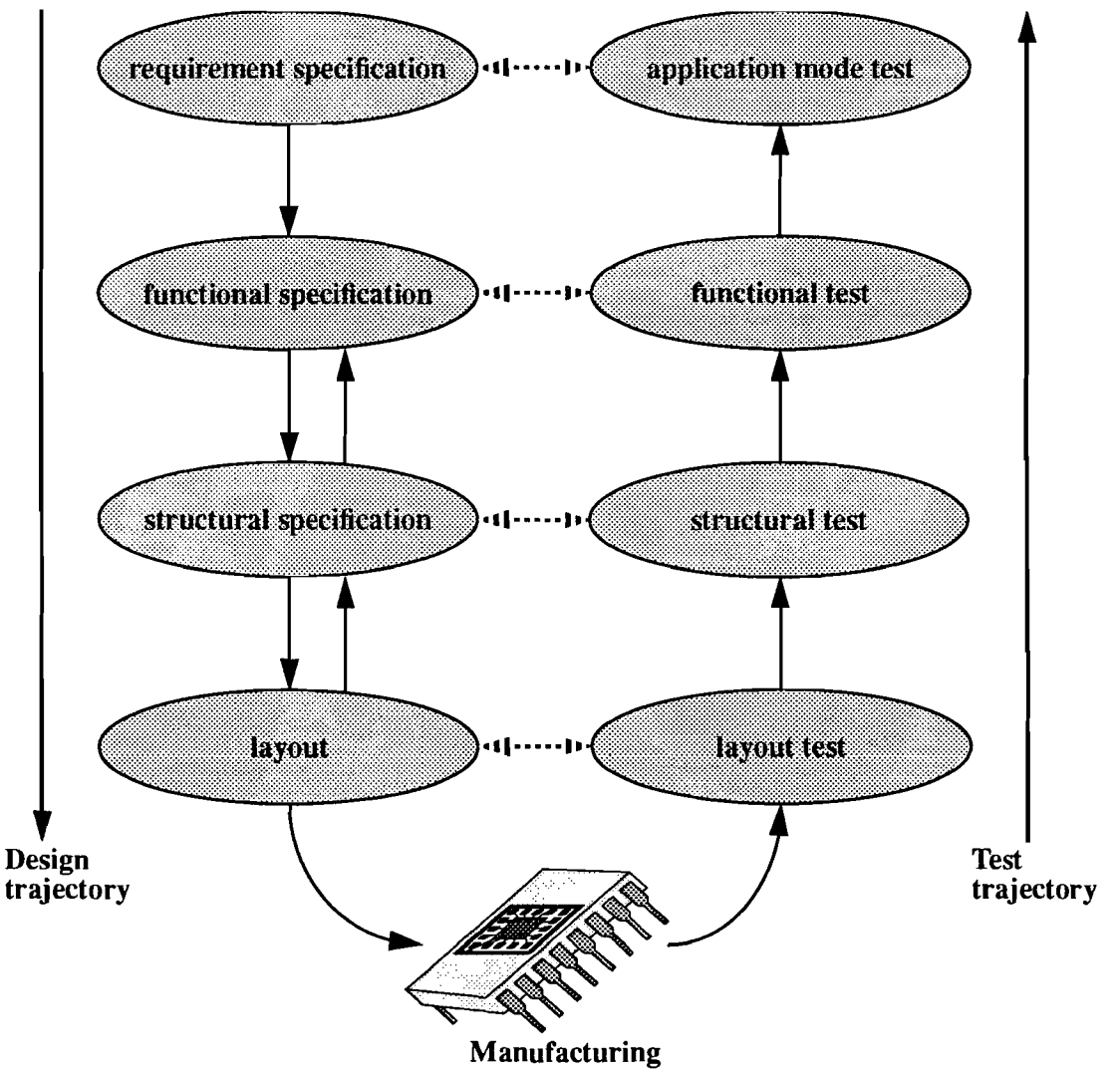


Figure 2-1: The IC design and test flow

2.2.3 Testing

To assure that the manufactured ICs perform the specified tasks, various tests are applied. Depending on the product development stage (e.g. prototype or mass-production), this test can be very extensive or just minimal to check the correctness of the production process. The three types of tests which can be distinguished are:

- Validation test: This test checks whether the device actually implements the specified functionality and can be applied in the complete system. Once the device has been shown to be functionally correct, it does not need to be validated again. The horizontal arrows in Figure 2-1 represent the validation processes.

- Characterization test: This test checks under which conditions the IC is able to perform according to the requirements. It includes checks on process variability, temperature variations, and power supply changes. This test is performed on a small sample size and only needs to be carried out once for a design and a given process.

- Production test: This test checks that no abnormal processing conditions or local disturbances occurred during the manufacturing process, causing a particular IC to malfunction. Because such a defect can occur on any device, each IC has to be tested for its correctness with respect to the processing.

Validation and characterization tests both are part of what is called *verification*. Production tests are not part of the verification trajectory!

Each of these three tests can be performed at various levels of abstraction, which are indicated in Figure 2-1. At the lowest level of abstraction, one should test whether the layout as specified by the designer is actually realized on silicon, bearing in mind certain process margins. Despite the direct relationship between a layout test and the detection of defects, this level of testing is only starting to be applied because of the relatively computation-intensive technique of fault list generation and test vector generation.

For production testing, the structural level is used in most situations. Because of the available fault models, which are an abstraction of the electrical impact of the defects on the corresponding level of design description, and corresponding efficient algorithms for test vector generation and evaluation, structural tests offer an excellent opportunity to derive a metric for the effectiveness of a test set (fault coverage). For the application of these structural tests, the scan technique often is used as simple controllability and observability improvement for the sequential circuit elements.

During the 80s and begin 90s, the stuck-at fault model has been the de-facto standard for a large part of the industry in classifying the test effectiveness. However, it should be remarked that the limited accuracy of the available fault models, especially the stuck-at fault model, in representing the behaviour of actual defects and the limitations of the static behaviour of a scan based test are becoming more evident and it is becoming more and more difficult to

meet the ever-increasing quality requirements. Additional and alternative test methods are considered to overcome these problems. These vary from voltage stress measurements to supply current measurements in the quiescent mode of a circuit (I_{DDQ} or I_{SSQ}).

Structural tests concentrate on finding spot defects and not on parametric defects. For example, as a result of a local processing disturbance, the signal propagation delay between two points on an IC might be increased, causing the IC to fail the functional specification, although the structure of the connections is not changed. Therefore, some tests also need to be carried out at the functional level.

On top of this level, it might be necessary to test the operation of the device in the way it is supposed to function in the system it is designed for. This is known as application mode test. Here the issue to be checked is not whether the IC implements a certain algorithm, but whether the implementation of that algorithm in combination with the other part of the system performs the expected tasks. In order to obtain enough controllability over the various conditions which might need to be varied also an application mode test is preferably applied on an automatic test equipment (ATE) instead of a bench implementing the application. However, this implies some requirements with respect to the availability of the right input signals and processing of the output signals [Bouwman, 95]. Application mode testing is generally only applied for validation and characterization purposes. For production testing this approach is too costly and will only be used if the quality demands of the customer are high.

The area of interest to us in this thesis is structural testing. Therefore, we look at this kind of testing more closely in the following section.

2.3 Structural test

The higher density of modern circuits results in an enormous set of possible fault cases. The main problem in structural testing is how to detect such faults, given the limited accessibility via the IC pins.

The requirements for a structural test are strict. The test should be fast because it is performed on every individual IC, and it must still have a high fault coverage since customers do not accept faulty ICs. Furthermore, it is important to be able to generate this test in a relative short time in order to prevent lengthy design times. It is generally believed that these requirements can only be met if already during the design phase the IC testability is taken into account. This is referred to as design for testability.

2.3.1 Design for Testability

Testability can be defined as follows [Bennetts, 84] : "An electronic circuit is testable if test-patterns can be generated, evaluated, and applied in such a way as to satisfy pre-defined levels of performance (v.g. detection, location, application) within a pre-defined cost budget and time scale".

Using this definition Design for Testability (DfT) can then be defined as the design effort that is specifically employed to ensure that a device is testable.

There are two important attributes related to testability, namely *controllability* and *observability*. Controllability is the ability to establish a specific signal value at each node in a circuit by setting values on the circuit's inputs. Observability is the ability to determine the signal value at any node in a circuit by observing its outputs.

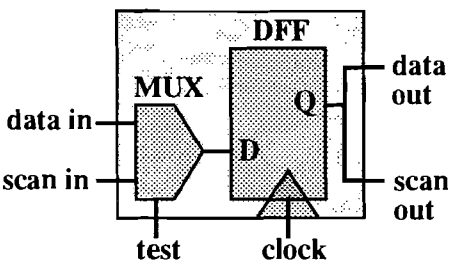


Figure 2-2: Scannable D-flipflop

When the scan test DfT technique is used during the design phase the testability of the design improves drastically by creating an extra test mode for the design. In normal mode, the design operates just as it is supposed to, according to its specification. In test mode, the memory elements will be connected into one or more scan paths. To achieve this, the memory elements must be replaced by versions that are equipped with such a test mode (this version is called scannable variant). In Figure 2-2 a scannable D-flipflop (DFF) is shown. The multiplexer (MUX) directs data either from data-in or scan-in to the D-input of the flipflop depending on the logic state of the test signal, which denotes if the scannable flipflop is used in test mode or in normal mode. In test mode the input of the DFF is directed from scan-in.

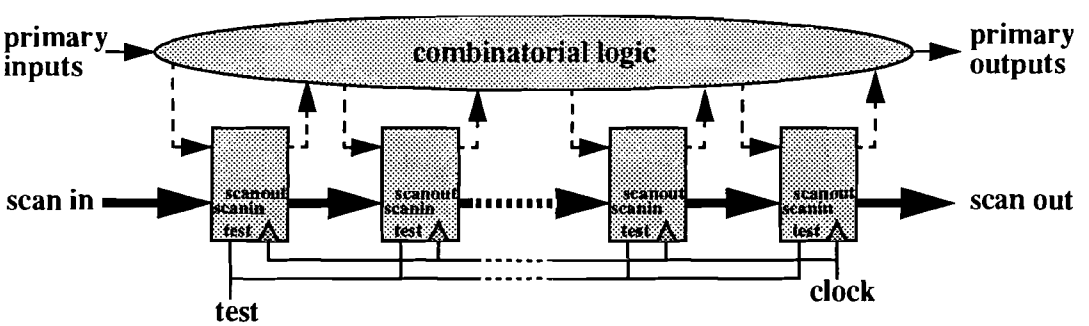


Figure 2-3: Scan path during test mode

When connecting scan-in and scan-out of different scannable flipflops, a potential *scan path* arises. A scan path is a shift register that only exists during test mode. Test patterns can be shifted in this register and applied to the rest of the circuit (controllability), which now only contains combinatorial logic, see Figure 2-3. After applying the test patterns during normal

mode, responses can be captured at the primary outputs of the circuit and the inputs of the memory elements. Switched back in test mode, captured responses in the memory elements can be shifted out of the scan path and gathered by the tester (observability). In this way, test pattern generation has only to be done for the combinatorial part of the circuit, this eases the process of structural testing.

2.3.2 Increasing fault coverage

The conventional methodology for testing digital CMOS circuits is the execution of a test sequence which exercises as many of the internal nodes and transistors as possible. A test pattern sequence should be able to detect, as a minimum, any stuck at 1 or stuck at 0 fault. On very complex VLSI devices, the development of the test pattern to achieve 100% fault coverage can be very tedious and extensive, resulting in many millions of vectors being generated. Even then, the level of fault coverage may not be sufficient to provide the degree of quality desired.

Quality level is a function of *total* fault coverage, of which the single stuck-at is simply an estimator. To maintain a given quality level, fault coverage must increase with increasing chip size (assuming the defect density of the fabrication process remains constant) [Maxwall, 92]. Since stuck-at tests by themselves are inadequate, the detection of other fault types needs to be quantified in order to obtain a good estimate of total fault coverage.

One method of getting higher total fault coverage is by adding functional test vectors to the structural test. As mentioned before signal propagation delay between two points on an IC might be increased, causing the IC to fail the functional specification, although the structure of the connections is not corrupted. In chapter 4, in which automatic test equipment is described, the term *functional testing* is used for both structural and functional test. From a tester point of view there is no difference.

Another method to obtain higher fault coverage is a test method based upon power supply current measurement. This method, also known as I_{DDQ} testing, is capable of detecting physical defects such as bridging, gate-oxide shorts, and spot defects in CMOS circuits. It is important to realize that at present time I_{DDQ} testing is supplemental to voltage testing. To verify functionality, voltage testing is still required. However, reliability can be significantly improved by I_{DDQ} testing.

3 I_{DDQ} testing

3.1 Introduction

Current testing is a method for enhancing the quality of IC tests by monitoring the level of power supply current. It is primarily applied to CMOS circuits where the quiescent power supply current, I_{DDQ}, in a defect-free device is very low. In the presence of a defect , I_{DDQ} can be orders of magnitude higher. Thus by monitoring the power supply current, it is possible to detect defects and achieve a higher device quality.

3.2 Basic concept of I_{DDQ} testing

A CMOS gate consists of an NMOS pulldown network and a complementary PMOS pullup network. In a fault-free situation for any given inputs, only one part conducts, connecting the output node to either the V_{DD} or the V_{SS} node. The gate output voltage is well defined at either logic level 1 or 0, and the circuit does not provide a conducting path from V_{DD} to V_{SS}. Thus in the fault-free situation, steady-state current in the circuit is negligible. The circuit contains only some junction leakage current. The magnitude of this leakage current is in the order of nA for VLSI and for all practical purposes it can be neglected.

In the presence of various physical defects, the steady-state current in a CMOS circuit no longer remains negligible. The magnitude of the power supply current increases in the presence of a fault. Thus, by monitoring the power supply current it may be determined whether or not a circuit contains a fault.

3.3 Advantages of current testing

IC manufacturing involves many critical steps such as metal depositions of a few thousand angstroms thick, sub-micron photolithography, and etching contact holes a few tenths of a micrometer wide. Small imperfections during these steps can cause randomly distributed defects in the IC such as shorts between metal lines or open contacts between the input signal and the polysilicon gate control. I_{DDQ} can be used to detect these defects using test vectors based upon other fault models than the stuck-at fault model.

3.3.1 Total observability

I_{DDQ} can also be used to detect faults that can be found with voltage testing, using shorter test vectors. This is a result of the *total observability* I_{DDQ} offers. The benefit of I_{DDQ} total observability above voltage testing will be shown by example.

In Figure 3-1 a test pattern 'A=1, B=1, C=0, D=1' is necessary to detect a short between output Z_1 and the power supply. The first two bits (A=1, B=1) of the test pattern are necessary to activate the defect so that a faulty behaviour is produced. In this example the test pattern is designed to produce a "0" on a non-defective Z_1 and power will force the output value to be a "1". A "1" in place of the expected "0" identifies a defect.

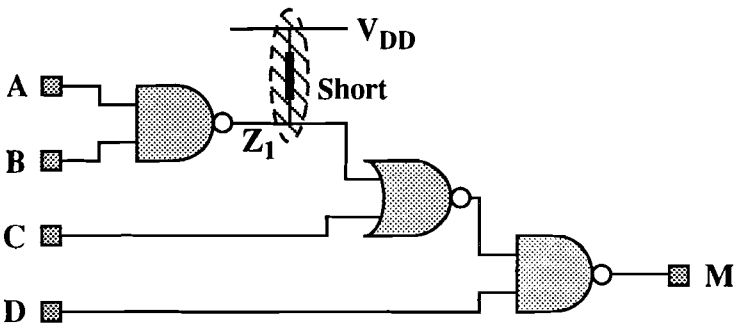


Figure 3-1: Test circuit with output Z_1 shorted to power

The remaining part of the test pattern (C=0, D=1) is necessary to propagate the defective behaviour to the primary output M, where a tester can observe the signal and compare it to the expected value. In the case of a sequential circuit, a sequence of test patterns is necessary to sensitize and propagate the defect.

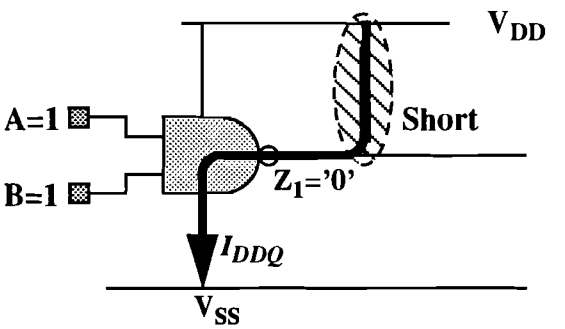


Figure 3-2: Fault detection with current testing

In current testing, the defective behaviour of a circuit is observed by measuring its power supply current rather than by propagating the fault to the primary output. In this case, a shorter test pattern (A=1, B=1) is required to detect the short shown in Figure 3-1. This is illustrated in Figure 3-2. A "0" on output Z_1 causes elevated quiescent current to flow from the power supply to ground in the presence of a short.

The faulty behaviour need not be propagated to an output since the power supply acts as an observation point on all gates simultaneously. This makes the test program generation task much simpler: only fault activation patterns need to be generated. It is found that fault activation is very similar to logic verification and that the simulation patterns used for verification are very effective at achieving high fault coverage with current testing.

3.3.2 Enhanced defect detection

Another advantage of current testing is its ability to detect defects that are not detected by the voltage detection technique that rely on stuck-at fault models. In stuck-at fault models, it is assumed that every defect causes the input or output of a gate to behave as if it is shorted to power or to ground. In the case of CMOS, many defects do not cause stuck-at behaviour. Figure 3-3 shows two examples of defects that are not characterized by stuck-at behaviour. i) short between two signal lines, and ii) an open contact on a FET.

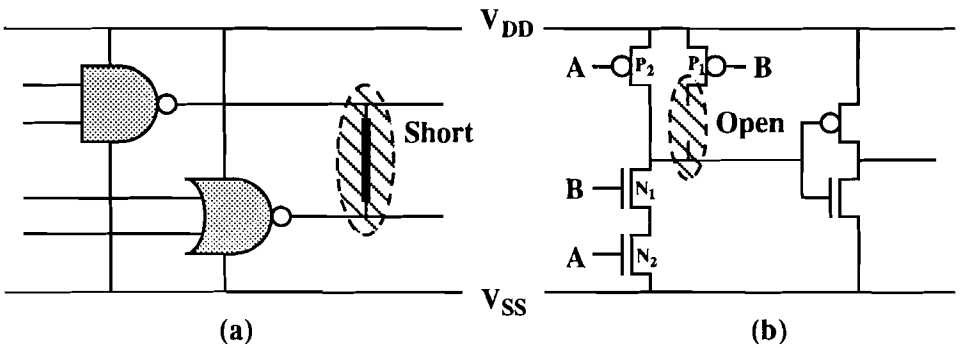


Figure 3-3: Examples of defects not covered by stuck-at testing

With current testing it is possible to detect many of the defects that escape stuck-at testing. In Figure 3-4 the tests for detecting the defects in Figure 3-3 are shown. In order to detect a short, complementary signal values are generated at the two ends of the suspected short. An elevated quiescent current will then flow through the power supply if a short exists.

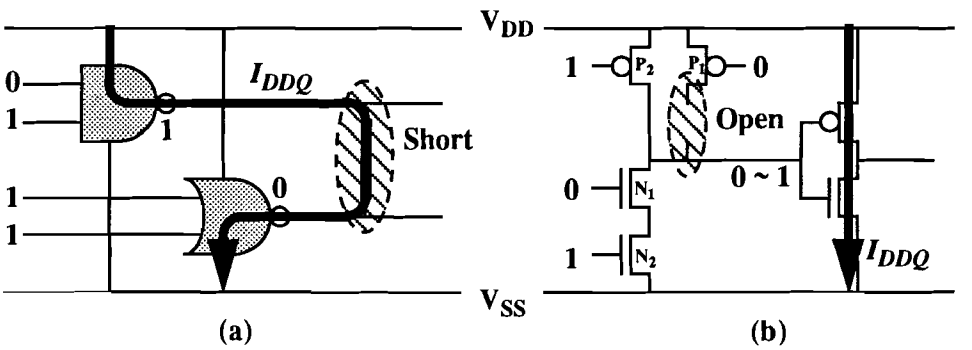


Figure 3-4: Detection of non-stuck-at defects with current testing

The detection of opens is more complicated. In this case a test pattern is applied that causes a net to float. In Figure 3-4 (b) this is achieved by applying a test pattern (A=1, B=0) that turns off the transistors P2 and N1. If transistor P1 is open due to a defect, the output floats. Initially the voltage on the output stays at a 1 or a 0 depending upon the charge stored on the output capacitance. However, if enough time elapses, leakage currents in reverse biased FET junctions cause the output voltage to drift to an intermediate value between logic 1 and 0. When that happens the logic gates controlled by the floating signal start to conduct D.C. current, causing an I_{DDQ} test failure. How much time should be allowed for sufficient I_{DDQ} to develop depends upon the capacitance on the node, the leakage current, and the size and threshold voltages of the FETs being driven by the floating node.

In addition to detecting shorts and opens, current test can also detect small leakage currents which may indicate long term reliability problems. For example, excessive gate leakage (current flowing through the gate oxide to substrate) may not cause an immediate fault, but it can lead to long term failure.

3.4 Limitations of current testing

Current testing is an 'inexpensive' and effective way to enhance the quality of CMOS ICs. However, it has several limitations that must be kept in mind. Two most important limitations are the signal to noise ratio and the test time.

To avoid failing good devices it is necessary to have a good signal to noise ratio. The signal is the D.C. current flow caused by a defect. If the defect is a short to power supply or a short between two nets, the signal would be of the order of several hundred microamperes or even a few milliamperes. However, some of the defects create a much smaller signal. The noise current is caused by the valid leakage current through the reverse biased junctions in each transistor on the IC. In addition, current transients caused by switching can have very long time constants, approaching several milliseconds. Any noise on the power supply or input signals can also cause current to flow through the device. For example, a 50 mV noise on the ground can cause 5 to 10 microamperes of noise in moderately dense ICs. As the number of transistors and wires on a chip increase, the noise continues to grow; at very high gate counts, the noise level is so high that it often masks defects.

The other limitation of current testing is the long measure time. Enough time should be allowed for i) the power supply transient currents to die down; and ii) the floating nodes to drift sufficiently to cause a current flow. For example, a floating node with 1 pF capacitance could take over 50 milliseconds to change its value from 5 to 3 Volts. Test economics limits the amount of time that can be spent on the number of measurements that can be made.

Selecting which vectors to perform I_{DDQ} testing has been a problem until recently. I_{DDQ} testing detects many common classes of process defects, such as bridges and opens, in a much more direct fashion than functional or scan test. However, the industry is accustomed to measuring fault coverage via the stuck-at fault. If I_{DDQ} testing is not based on a fault model then testing many tens of thousands of vectors with a functional test maybe the only alterna-

tive. If the industry could agree to measure fault coverage for I_{DDQ} on the basis of a common model, it would be possibly to reduce the number of I_{DDQ} vectors for VLSI ICs to a few thousand possible a few hundred.

How many vectors are necessary for efficient I_{DDQ} testing is a serious question. However, it is a problem surrounded by controversy. One fact that is widely accepted is that current Automatic Test Equipment (ATE) systems are too slow, and that ten to twenty I_{DDQ} vectors are the limit without adaptations. So I_{DDQ} can only be used as a supplementary test with today's instrumentation. If the industry had an accepted fault model, it is possible that I_{DDQ} vector selection could be minimized to a few hundred vectors. In combination with faster measurement hardware this could lead to acceptance of I_{DDQ} as a general test method.

4 Automatic Test Equipment

4.1 Introduction

While testing, stimuli data has to be applied to the Device Under Test (DUT) and resulting responses have to be captured and compared with expected data. The equipment that translates the stimuli data into voltages that are set on the right times on the right pins and that observes output pins to compare with expected data is called Automatic Test Equipment (ATE).

4.2 What ATE is

ATE is computer-controlled equipment that is used to verify certain parameters of a DUT and to identify defects that may be present in the DUT. It may, in its most general form, be block diagrammed as shown in Figure 4-1.

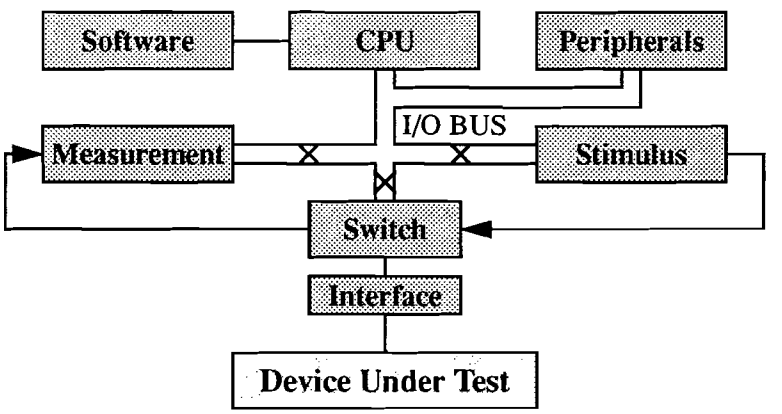


Figure 4-1: Block diagram of generic ATE system

At the heart of the system is the "control section". The control section consists of the computer (CPU), its peripherals, and its software. The computer executes application-specific software that directs the stimulus, measurement, and switching resources of the system in a manner that will yield the desired test results and data.

The stimulus section provides vectors to the DUT. The measurement section evaluates the DUT's response to the vectors from the stimulus section. The switching section may be thought of as the programmable path network between the stimulus and the measurement section. The control part then operates upon the information obtained from the measurement section to determine the next course of action (e.g., abort the test, issue a failure message, or issue a "test pass" message).

The ATE system is interfaced to the DUT via an interface adapter which is often DUT-specific. This interface adapter may be as simple as wires between the ATE system and the DUT, or it may contain extensive electronics for level shifting, signal translation, and special signal conditioning.

Typical stimulus devices for electronics ATE for large-scale integration (LSI) and very-large-scale integration (VLSI) integrated circuits and the printed circuit board assemblies (PCBAs) include programmable power supply units (PSUs), signal generators, and data driver pins. Typical measurement devices include parametric measuring units (PMUs), digital multimeters, timer-counters, and data receiver pins. Typical switching systems may be constructed of either relays or solid-state logic circuits, depending on the levels and types of signals to be routed. The stimulus, measurement, and switching devices and circuitry may be configured as individual items interfaced to the control section central processor unit, or may be designed as integrated, multifunction electronic assemblies.

Pass/fail parameters are typically programmed into the system when the application test program is written. This situation is illustrated in Figure 4-2. Immediately prior to test program execution, digital drive levels are set on the stimulus pins and digital receive thresholds are set on the measurement pins. Then the specific functional "1"s and "0"s are driven into the DUT and sensed from the DUT by the tester electronics. On the measurement side, if a signal exceeds the logic "1" threshold, it is assumed to be valid logic "1". If it is below the set logic "0" threshold, it is assumed to be logic "0". If the signal is between the two thresholds, it is deemed to be a defective level and indicates that the DUT is not functioning correctly.

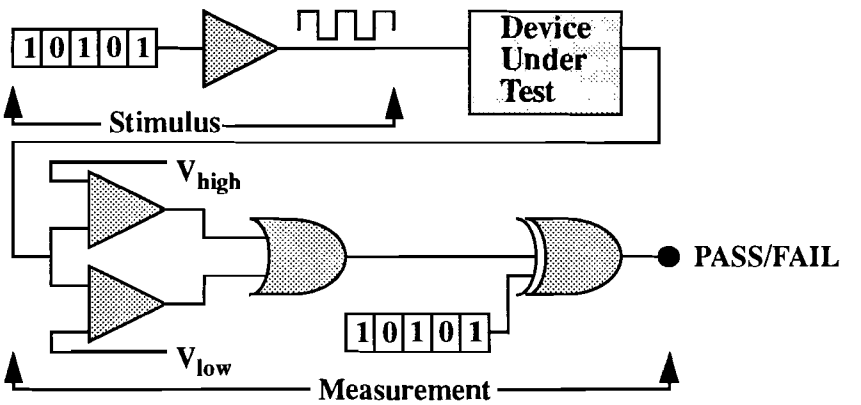


Figure 4-2: Schematic diagram of digital pin electronics

The only other indication of non functionality is if a "1" is received when a "0" was expected, or vica versa. Some testers allow for the programming of a time limit by which the response must be received, and others allow for the sensing of driver pins to ensure that the stimulus signal was actually able to drive the DUT. Regardless of the level of sophistication of the electronics and the software, if the tester does not provide specific parametric values as part of the test results, it is considered a functional tester.

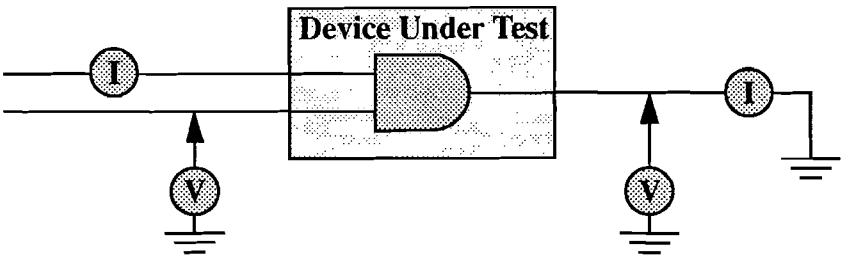


Figure 4-3: Circuit diagram of DC parametric measurement

Adding parametrics involves adding enough hardware and software to provide actual measurements in engineering units (e.g. volts, amps, and seconds).The term DC parametrics (see Figure 4-3) refers to such parameters as input current, output voltage, and supply current. The term AC parametrics refers to such parameters as rise time, fall time and propagation delay (e.g. time between input stimulus and output response). It is possible for a functional ATE to provide go/no-go (pass/fail) indication of parameters without providing actual measurements in parametric units. For DC parametric approximations, one way that this is accomplished is by running the ATE with drive levels, thresholds, and loading conditions set as close as possible to the parametric limits of the DUT.

4.3 Testing overview

First goal with testing is to insure that the DUT operates according to its truth table. Figure 4-4 shows the general structure for testing logic devices.

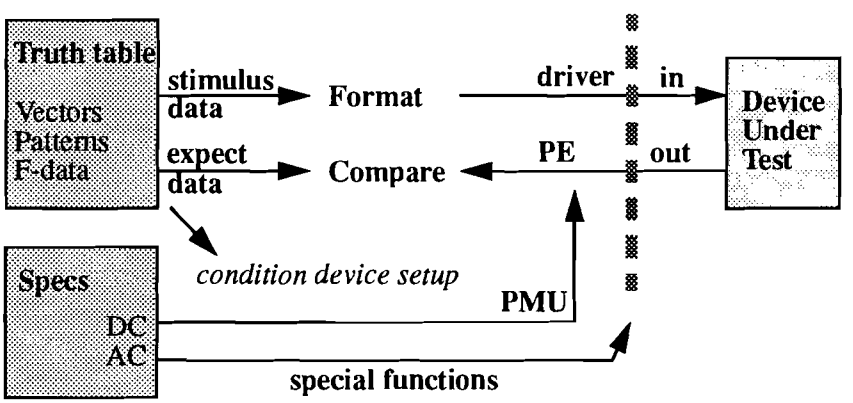


Figure 4-4: Block diagram of testing process

The truth table is loaded into the tester in the form of patterns (vectors or F-data). To get the data to the device, stimulus data is output from the pattern memory. It is passed through the formatter, which combines the data with timing edges to form a waveform. The waveform is passed to a driver, which converts the data levels to programmed voltage values. The voltage values called "input references", are supplied by the test engineer in the test program. The total waveform is passed to the device through the interface. The interface consists of a load-

board, which connects to the tester head, and a socket which is wired to the loadboard. The device respond with an output, which is passed through the loadboard to a comparator in the testhead. Expected data is converted by output voltage references and compared with device output. Naturally there is a driver and comparator for each pin. The pattern memory is also used to pass a sequence of patterns to condition the device to a state where the DC parametric measuring unit can be connected, under software control, to various pins for making voltage and current measurements.

4.4 Functional testing

Functional testing refers to the process of accessing the normal input/output (I/O) interface of the DUT (its package pins) and providing stimulus patterns and measurement verifications that the DUT is functioning. Functional testing relies on the assumption that if the DUT functions, no process-induced problems exist.

4.4.1 Truth table

Logic devices are tested by two types of testers, Golden Device and truth table (stored response). Golden Device testers compare the DUT with a known good device. They are only used during production or incoming inspection. The most common tester is the stored response or truth table tester. It stores the truth table for a device in its memory, formats the data, and applies the data to the device. One line of a truth table is called pattern or vector, they are applied to the device one at a time. Each vector or pattern consists of stimuli and expect data. Vectors are stored in the high-speed pattern memory of the tester.

The test program loads vectors into the pattern memory and applies its pattern data to the device. Each test program will call several vector loads to test the different functions of the DUT. The pattern data must be formatted before it is passed on to the DUT. This means that it must have timing and voltages assigned to it.

4.4.2 Creating stimuli data

Truth table / pattern data alone is a poor method to stimulate the device, because of the weak control of timing. The pattern data must be formatted and passed to the device through the tester driver. This is illustrated in Figure 4-5. The formatter couples the pattern data with timing generators (TGs) or timing edges. At the device, data appears as a timing diagram.

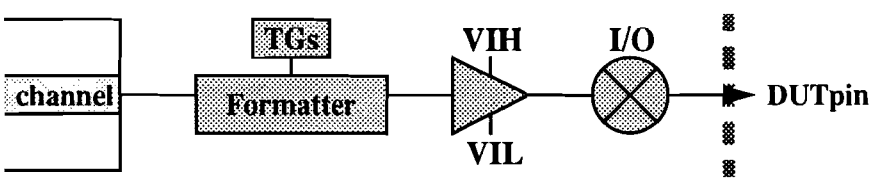


Figure 4-5: Schematic of tester hardware which creates input waveforms

Each cycle begins at time zero, T_0, with respect to the device, and lasts for a certain length of time specified in the test program. Most testers have a number of timing generators or clocks that are shared by all the pins. Figure 4-6 illustrates the timing parameters used in testing.

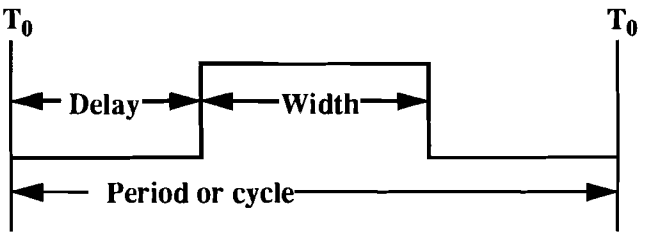


Figure 4-6: Timing parameters used in testing

A waveform is the output of the formatter. Waveforms are individually specified for each pin. The common waveforms are shown in Figure 4-7. Some testers define timing with edges (or phases).

Common edge definitions would be:

AEDGE: Data transition edge
BEDGE: Leading edge of pulse
CEDGE: Trailing edge of pulse

The formatter couples the edges with the pattern data to form the standard waveforms.

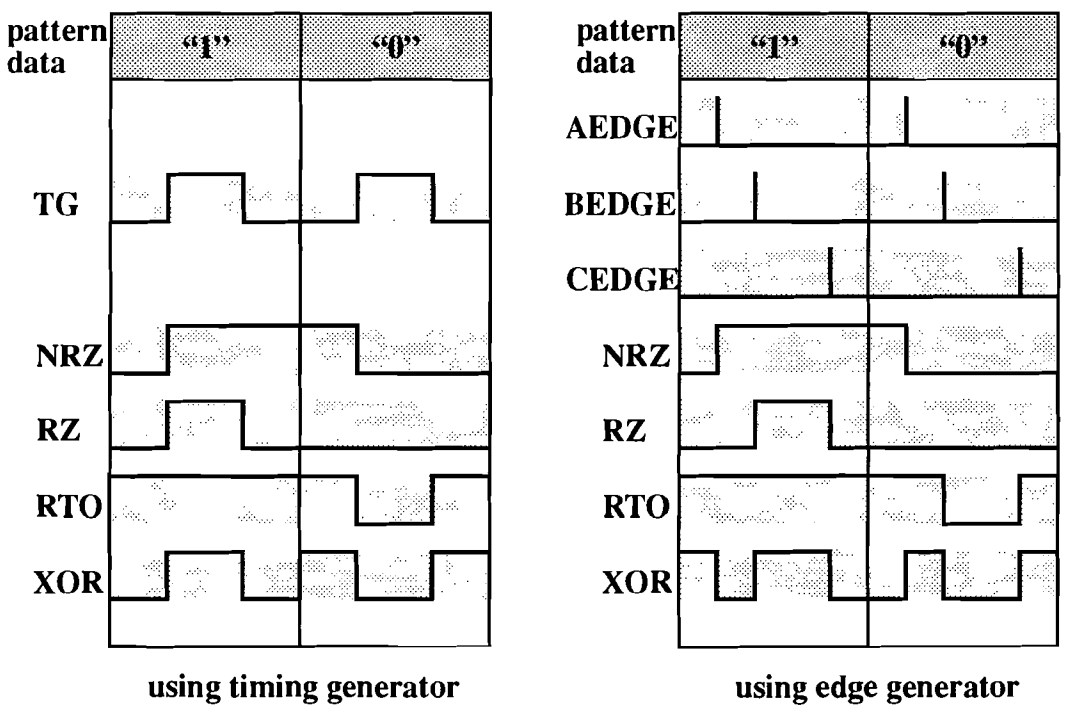


Figure 4-7: Common input waveforms created by formatter

Depending on the tester, the flexibility of its timing generators to create waveforms will vary. The common waveforms illustrated in Figure 4-7 have the following properties:

NRZ: Nonreturn to zero or straight data. When coupled with a TG, the data transition is delayed from T_0, but the data level remains for the length of a complete period.

RZ: Return to zero. When pattern data is "1", a pulse equal to the assigned TG is created. The level remains low for pattern data "0".

RTO: Return to one. When pattern data is "0", a negative pulse equal to the assigned TG is created. The level remains high for pattern data "1".

XOR: Exclusive OR (surrounded by complement). Positive pulse on pattern data "1". Negative pulse on pattern data "0".

Although the above four waveforms are used in the majority of applications, the possible waveforms available on modern VLSI testers will not be limited to them. The formatter creates the waveform and passes it to the driver, which converts the logical levels to voltages.

4.4.3 Comparing DUT output against expected data

A comparator is used to compare the output of the DUT with the expected pattern data. A schematic of the comparator is given in Figure 4-8. The DUT output is passed to the comparator through the loadboard connector. The pattern data (expected data) selects the high or low level of the output reference assigned to that pin. There is a choice of at least two output reference pairs that can be assigned to each comparator. The level is compared with the DUT output and the result is passed to the next stage of the pin electronics for pass/fail determination.

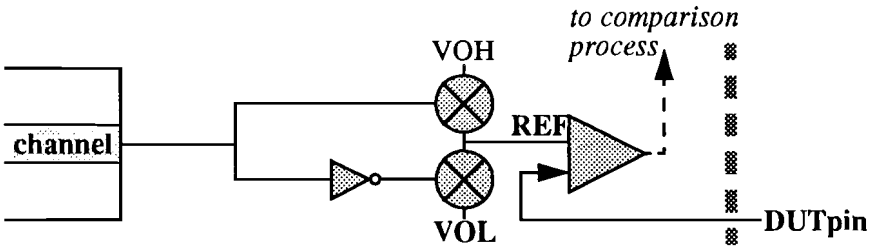


Figure 4-8: Schematic of tester comparator hardware

A failure is recorded only if the comparison is incorrect at a specified time as determined by a strobe. There are two types of strobes, edge and window. With edge strobe an instantaneous latch of comparison results occurs. Failure is recorded only at a specified instant in the cycle. Using window strobe, the comparator result is observed during an interval. Any failure of the comparison during the interval is a failure; i.e., the DUT output must be at the correct level during the entire interval.

For various reasons it may be desirable to mask the failure results for certain cycles or pins. Although the comparator is always attached to the pin, the comparator results are not monitored for input pins. Also during input cycles of I/O pins the comparator is disabled. The user must be given the option of masking the failure results for any cycle, known as a "don't care" cycle. Thus, to have complete control, the following has to be possible on a cycle-by-cycle basis:

1. Switch driver from driving to Hi-Z.
2. Enable and disable comparator results.

This is accomplished on modern testers by embedding the I/O control and masking within the pattern data. The pattern data for each pin will have a minimum of five states as shown in Table 4-1.

Table 4-1: Minimum tester states

Pattern data	Driver state	Comparator state
1	drive one	disabled
0	drive zero	disabled
H	Hi-Z	compare against high level
L	Hi-Z	compare against low level
X	Hi-Z	disabled

The results of the comparator can be masked for an input-only pin by not assigning a strobe to that pin. This is obvious from the complete diagram of the comparison process, given in Figure 4-9.

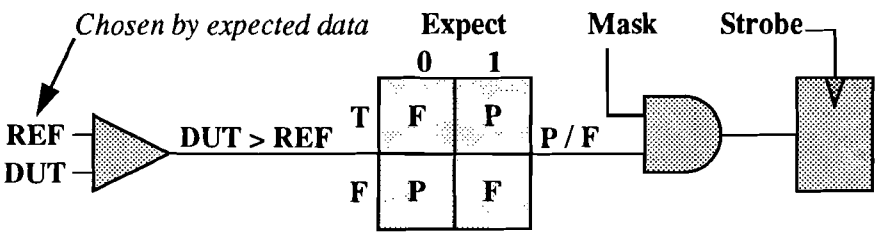


Figure 4-9: Diagram of output to expected data comparison process

The expect data selects an output reference level, which is compared with the DUT output. The comparison results are passed through some logic for decoding, as represented by the truth table after the comparator. Depending on the expect data, a pass or fail is passed on as a

"0" or "1". The failure can be masked by the pattern data. The pass, "0" or fail, "1" is latched into the results register by the strobe. Notice that the default condition in the results register is a "0"; i.e., a pass is assumed unless otherwise shown.

Finally a mechanical relay must be closed to connect the pin electronics to the device. Different methods are used to close the relay in different testers. Some require at least one driver register to have a "1" in it for that pin; some require a direct closing statement; and others close it whenever a pin is defined. The point is that it can be controlled.

5 Quality Test Action Group (QTAG)

5.1 Introduction

QTAG was proposed at a public meeting of the IC test engineering community at the International Test Conference in 1993 (ITC 93). The need for QTAG has arisen from the rapidly rising quality demands that the cutsomers of CMOS ICs are making on IC vendors. In general, the experience of the IC vendors has been that significant improvements in overall realistic fault coverage, using the classical stuck-at fault model, cannot be expected. Adopting more advanced fault models based on realistic defect probabilities leads the test engineering department to adopting I_{DDQ} based test methods for advanced CMOS devices [Baker, 90].

Experience over a number of years with I_{DDQ} testing within Philips Semiconductors and reports from development centres in the industry indicate that I_{DDQ} testing has its drawbacks. Measurement of a significant number of I_{DDQ} vectors is excessively time consuming with conventional automatic test equipment (ATE) systems. In the past ATE vendors have been pressed to improve the performance of the ATE based I_{DDQ} facilities, but in general the facilities provided have not been widely used. For some CMOS products Philips has developed and maintained special test systems for I_{DDQ} testing. Such a discrepancy between the needs for I_{DDQ} testing and the capabilities of the ATE systems is heading for a "finger-pointing" crisis between the test engineering departments and the ATE vendors. Action is needed to avoid this crisis. QTAG's aim is to develop a de-facto standard for test fixture based I_{DDQ} monitors. The standard should allow the monitor vendors to develop innovative products with the functionality that will create a small, but active market-place for their monitors.

5.2 Goals of QTAG

The original proposals for QTAG made at ITC 93, had the aim of focusing the attention of the industry on the problem of I_{DDQ} production testing. The proposal was made to the industry that the solution was to be found in the so called *"Little-Foot"* monitors.

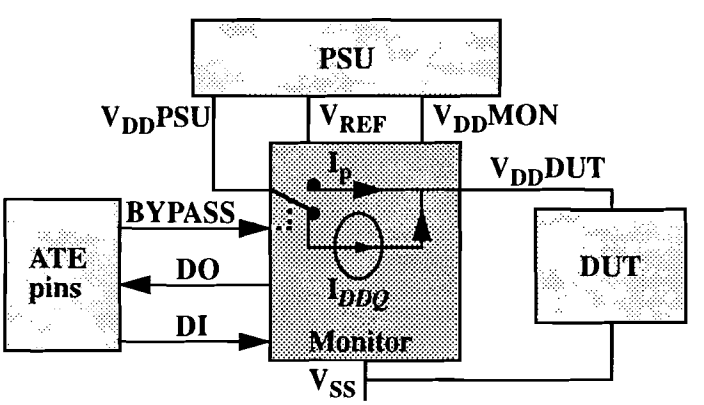


Figure 5-1: QTAG I_{DDQ} monitor with bypass option

These monitors are based on the technology developed for Built-In Current (BIC) sensors, see Figure 5-1, but packaged in such a manner as to be used on any test fixture. Two major modes are provided, the normal I_{DDQ} current monitor mode and the bypass mode. If bypass mode is activated a short circuit resistance is provided between V_{DD}PSU and V_{DD}DUT. Measurement during this period is not possible. In monitor mode measurements are made. The idea of the *Little-Foot* was that the physical foot-print on the test fixture was so small that it could even be used on probe-cards, see Figure 5-2. This requirement for small physical size, demands that the pin count be small (< 10 pins). An acceptable package is the SO-8, demanding less than 0.5cm^2 with SMD decoupling capacitors.

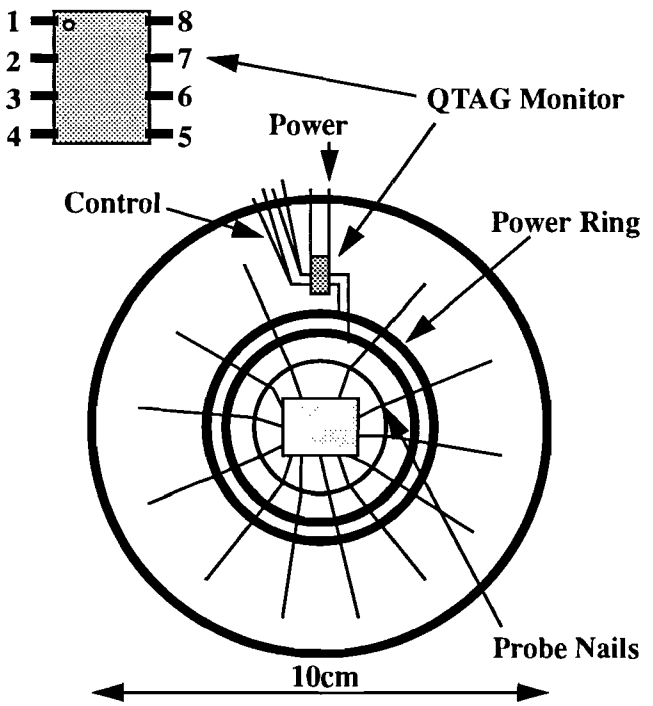


Figure 5-2: QTAG monitor on a probe card

Given that the monitor is to be driven by an ATE system virtually any digital interface based on three or four pins can be created. Originally, a simple two mode interface was proposed at ITC 93. However, it is better to allow a more complex interface to be defined which gives the capability to control more monitors with the same pins and permits much more functionality to be built in to future monitors [Hales, 94a]. At the same time the standard has to be realistic about the capabilities of monitors that could be designed in a reasonable timescale.

After ITC 93 a second class of monitor was defined. These are the so-called *Big-Foot* monitors. This class of monitor, probably not based on BIC sensor technology, requires a significantly larger area on the test fixture, so would not be as useful in production testing. However, given the known performance of such monitors, and the need to provide a measurement capability for engineering applications it was decided to cover them within the standard as a separate class. The idea being that such monitors could also be driven by the same ATE driver software.

A set of goals of QTAG was defined by Keith Baker of Philips Research and Chuck Hawkins of the University of New Mexico. These goals were re-stated in the Philips proposals for QTAG [Baker, 93] and have been reproduced below:

- 1. A definition of a "minimal-pin" configuration package for a monitor whose specific design may differ for any manufacturer. That such a monitor allows any digital test system to drive and receive information appropriate for I_{DDQ}/ I_{SSQ} testing.

- 2. A defined pin configuration that demands packaging in a minimal number of package types to cover most applications needs.

- 3. A standard ATE interface definition and physical form that allows all common test fixtures to be used: DUT board, probe-card and contactors.

- 4. A standard that defines both VDD and VSS line monitor configurations.

- 5. A standard that would recommend or define multiple monitors and multiple power pin configurations.

- 6. Definition of a Monitor Description Format (MDF) that would allow different monitor implementations to be easily driven by ATE hardware and software.

- 7. A standard that allows tester selection of a variable I_{DDQ}/I_{SSQ} threshold.

- 8. A standard that allows robust physical construction and reliable operation in a stressful Electro Static Discharge (ESD) environment.

5.3 QTAG monitor classes

In its proposal, Philips developed the original goals of QTAG into a structure with three classes of monitor. Details of the three classes are given in Table 5-1. Class 1 would be the original Little-Foot monitors. These would be tightly standardized monitors which would be the central thrust of the QTAG activity. From the specification of Class 1 the MDF would be created. The other two classes form the Big-Foot monitor types.

Progress during the Monitor Standard Definition phase of QTAG has been very uneven for the two different types of monitors. For the Little-Foot monitors, the semiconductor vendors have produced a very clear definition of the function and physical format of the monitor. For the Big-Foot monitors, the progress has been less rapid because of the lack of a clear champion for the concept among the semiconductor vendors.

Table 5-1: Overview of QTAG Monitor Classes

Class	Template cm	Function	Pins	Packages type
1	0.8 x 1	production	8	SO-8 SO-8L SO-14
2	2 x 2	production engineering	24	any
3	4 x 4	engineering	24	any

5.3.1 Proposals for Little-Foot monitors: Class 1

Texas instruments (TI) also made a proposal to QTAG in a much more detailed form than Philips. This developed the concept of the class 1 monitor in a definitive architecture. The TI proposal built on the original *Little-Foot* concept addressing many of its shortcomings. Main issues are using multiple monitors, providing a truly digital interface, giving a measurement option etc. [Hales, 94a]. These issues were originally not addressed by Philips [Baker, 93].

After the completion of the Monitor Standard Definition phase it is clear that the semiconductor vendors proposals for class 1 monitors give this class a leading role in development of the standard.

After bi-lateral discussions between Philips and TI, a common proposal based on the umbrella format from Philips and the class 1 specific proposal from TI has been put forward [Hales, 94b]. Philips would then develop a prototype QTAG class 1 monitor on the basis of this specification. This prototype would be a proof of concept study for QTAG [Baker, 94b].

A second European group, Alcatel Bell-Telephone, Technical Highschool of Ostend and the University of Hull, have also supported the development of the class-1 monitor by the development of a I_{DDQ} monitor based on discrete components. However, the pinning and functionality would be compatible with the general proposal for the class-1 [Manhoeve, 94].

5.3.2 Proposals for Big-Foot monitors: Class 2/3

None of the major semi-conductor vendors has made more detailed proposals for class 2 or 3 monitors. Given the lack of interest in these classes from those parts of the industry that could create a demand for a marketable product, the most direct solution would be to scrap them from the standard. Further enquiries reveal that there are groups developing such monitors, however, they have not found the time to fully participate in QTAG.

The Quic-Mon circuit, originally developed by Sandia Labs but further developed by one of the original authors with Philips Semiconductors, falls very clearly into the *Big-Foot* category. This monitor is without doubt the most popular type published todate and QTAG must ensure that users of this monitor are not excluded from support on ATE systems [Wallquist, 93].

5.4 QTAG monitor types

The QTAG monitor classes are based on pin count, function and physical size. The monitors can also be distinguished in capability. This is done by the QTAG monitor type definitions. In the next subsections the two basic types of monitor are discussed:

- Threshold: return a simple pass-fail, or analog value
- Measurement: returns a digital value

Current monitors can be divided in semi-digital and fully-digital versions. Semi-digital monitors are controlled by a mixture of digital and analog pins. These monitors are the majority of QTAG monitors in development at this time. Fully digital monitors are much simpler to interface to a digital tester because they have only digital control pins. Moreover, the binary control codes for digital monitors must be inserted in the test vectors of the DUT, which is an extra step. Semi-digital monitors do not need those codes in the test vectors. Digital monitors have an internal architecture that must either be standardized or defined within the MDF to allow the ATE to control the monitor. Also a distinction can be made in I_{DDQ} and I_{SSQ} monitors. The difference between these two types is the current observation point. In case of I_{DDQ} the measure pin will be connected to the DUTs V_{DD} pins, whereas with I_{SSQ} the measure pin will be connected to the DUTs V_{SS} pins.

5.4.1 Threshold monitors

The threshold monitor is the simplest of the QTAG monitors. In case of a digital output, it simply tests if the quiescent current (I_{DDQ} / I_{SSQ}) value is above or below a user-specified or fixed threshold value. The result of this comparison is returned to the tester as a binary pass/fail value (DO output). When multiple monitors are chained this signal will be forwarded by a chain of DI and DO pins. In case of an analog output value, this will be directed to the pin electronics comparator, which requires the digital tester to determine the pass or fail by comparing the input voltage to a known reference voltage set on the pin electronics comparator.

5.4.2 Measurement monitors

Measurement monitors provide either a parallel or serial digital word to one or more test system pin electronics comparators. In case of serial data, timing of the data is determined via a separate digital clock pin on the monitor driven by a pin electronics driver. With analogue testers or by using an ADC on the test fixture it is possible to create a measurement capability on threshold monitors with an analog output. When using an analog threshold monitor in this way it is called a measurement monitor.

5.5 QTAG monitor pins

The standard pins of a QTAG I_{DDQ} monitor are listed below in Table 5-2. The first three pins are power supply connections to the tester's power supply, and the fourth pin is the VDD supply to the DUT. The other four pins form the interface which the tester uses to communicate with the QTAG monitor. The MODE, CLK and DI pins are outputs from the tester to the QTAG monitor, and the DO pin is an output from the monitor back to the tester. Multiple monitors can be connected by chaining together their DI and DO pins and connecting the MODE and CLK outputs from the tester to the MODE and CLK input pins of all of the QTAG monitors.

Table 5-2: QTAG I_{DDQ} monitor pins

Pin	Description
VDD_MON	VDD supply for the QTAG monitor
VSS	VSS supply for the QTAG monitor
VDD_PSU	VDD supply for the DUT from the tester
VDD_DUT	VDD output from the QTAG monitor to the DUT
MODE	Mode select
CLK	Clock
DI	Data input
DO	Data output

5.6 Monitor Description Format (MDF) requirements

One of the QTAG goals is definition of a Monitor Description Format. At the initial meeting of QTAG the concept of the MDF was outlined. MDF was needed to allow the monitor to be described in a machine and human readable format, so that different monitors could be used with different test systems, and would allow the test engineer to efficiently utilize the monitor in the application.

It was a stated goal that the MDF should be so broadly formulated as to allow monitors not fully compliant with the QTAG standard to be used. This may seem strange at first. However, the QTAG document [Baker, 94a] makes it clear that this is essential. For example, the QTAG monitor proposal will allow monitors to be daisy-chained to allow two or more monitors to be used in conjunction. However, this demands that a pin is available as digital input pin, DI, but for so-called semi-digital monitors in stand-alone applications this would waste a valuable monitor pin. So flexibility is needed to provide a standard that maximizes the potential of the individual monitors.

The semantics of the MDF language are based on the resources provided by a simple digital tester, i.e. pin memories, pin driver, pin comparators, PSU, PMU, and a test program executed on a computer. Each of these resources is assumed to be under control of the test program. A test program is a computer program executed by the control computer of the test system. It is assumed that the test program has access to a set of standard routines to control the monitor. Furthermore it is assumed the files of test vectors loaded by the test program into the pin memories and pin timing generators have been so modified by an external program that all the control, triggering and timing of the monitor and DUT are correct.

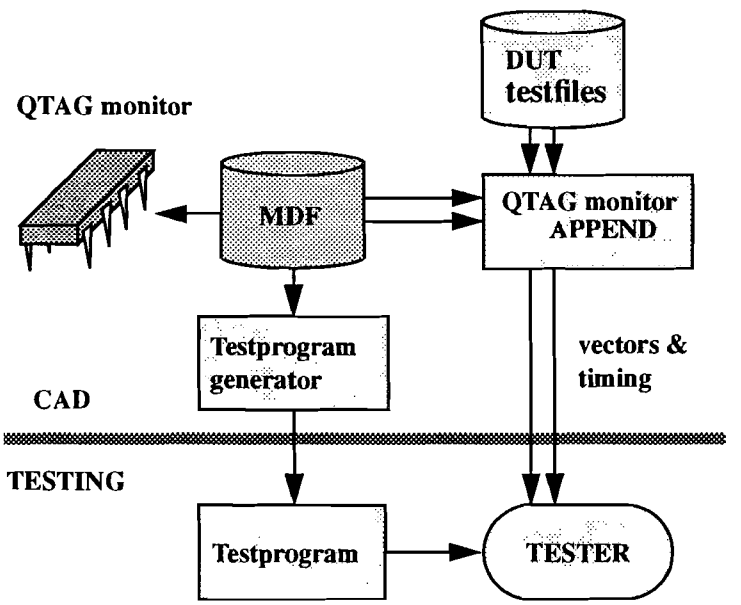


Figure 5-3: QTAG Design-test flow

Using the semantic model described above, the MDF has to contain the information for two functions, as shown in Figure 5-3. First the MDF must contain the information needed to allow a filter function called Append, to modify the Computer Aided Design (CAD) generated test data to include the monitor timing and vectors. Second, the MDF must provide input to an automatic test program generator to create the correct test programs to control the tester resources.

MDF is not a general purpose hardware description language. It is intended solely as a means of describing the key aspects of the implementation of I_{DDQ}/I_{SSQ} monitors used in conjunction with ATE systems. MDF is not a simulation model.

Features Described by MDF:

- Pin declarations and pin definitions for monitors
- Definition of architecture of digital monitors
- Definitions of monitor actions
- View of monitor for test fixture

Features not described by MDF:

- How chains of monitors are linked on a test fixture
- Test programming or test functions
- Input format from CAD tools for ATE

The MDF is best served by a simple format specifically created for the application. So that the general behaviour of the implementation is defined in the standard and only the device specifics are provided in MDF. In this the role model of BSDL used by IEEE std 1149.1, which specifically excludes extraneous details is followed.

In chapter 7 MDF is defined according to the requirements stated in this chapter and the needs that followed from the stock taking of present available I_{DDQ} monitors and detailed proposals for future development in chapter 6. The architecture of digital monitors has not been defined in MDF so far, due to the lack of a standard digital interface. This part of MDF will be future work. The MDF defined yet covers all QTAG monitors known and expected in the future, including the digital monitors without specific interface information needed to control this type of monitor.

6 I_{DDQ} monitors

6.1 Introduction

Developments in monitor design have a great deal of impact on the semantics and syntax of MDF. At this time, the only monitors actually in development are simple types that use only a semi-digital interface. This requires that the ATE system must be able to supply analogue signals or accept analog input signals. In this chapter first two measurement principles are described in section 6.2. In the following sections an overview is given of the current monitors developed and published in the literature. Each monitor will be described, mentioning details that are used while defining MDF. This implies that the controlling part of the monitors will be main issue. In section 6.8 a fully digital interface that was proposed to the QTAG-group is discussed. Below is a short summary of the monitors presently available:

- **QuiC-Mon v3.2**
  Developed by the university of New Mexico and Sandia Labs, this is one of the best defined I_{DDQ} monitors so far published in the literature. It is a complex monitor that provides a digital measurement capability to the test system. It is suited to the task of device characterization and analysis.

- **QuiC-Mon v5.0**
  A new version of the original QuiC-Mon monitor, further developed by one of the original authors with Philips Semiconductors. This is a simpler monitor, that can be used for both I_{DDQ}/I_{SSQ}, but provides an analogue value to evaluate as pass/fail.

- **OCIMU**
  Developed by Alcatel-Bell and Technical School Ostende to demonstrate the feasibility of QTAG monitors for I_{DDQ} testing. It is capable of performing a relatively high speed (measurement time can be less than 30 µs for a 2µF load) current measurement especially when driving a high capacitive load (several µF). It provides a digital pass/fail to the test system.

- **IDUNA-2**
  A demonstration QTAG class 1 monitor developed by Philips Research and Lancaster University in Application Specific Integrated Circuit (ASIC) form. It provides a digital pass/fail to the tester based on a fixed internal threshold current, that can be altered through a analog pin.

- **LTX I_{DDQ} monitor**
  This monitor is a stand alone option of a LTX test system. It has been designed to integrate into the current test systems. The LTX trillium test head is modified to accept the I_{DDQ} monitor. It provides a digital measurement capability to the test system. Internal threshold can be set by a binary value on four digital pins.

6.2 Monitor principles

The problem of measuring I_{DDQ} current is basically measuring the current flowing into the DUT after all transients have settled to a steady state. It is important that the monitoring device should not influence the DUT. This implies that the I_{DDQ} monitor should be transparent at the moment the logic state of the circuit changes. Finally the monitor should be placed as close as physically possible to the DUT (actually on the loadboard) to minimize environmental influences. Especially since during the test of an IC a lot of signal activity takes place in the testhead to which the loadboard is attached. Test signals with rising and falling times less than 1 ns are not uncommon, this generates a lot of high frequency switching noise which can disturb other signals and thus create a hostile signal environment. As a result and because of the noise accurate measurements of voltage levels below 1 mV are not feasible, connection wires are to be kept as short as possible.

To measure DC currents two basic principles can be used. The first one is based on a current to voltage conversion by placing a resistive sensing element in the current path and applying Ohm's law. The second principle is based upon a voltage to time conversion by measuring the time needed to let the voltage across a given capacitor drop by a predefined amount. This method is based upon the discharging of the capacitor over a load resistance when it is disconnected from the supply voltage, this is known as Keating-Meyer principle [Keating;Meyer,87].

The main drawback of the first method is that the introduction of a resistive element in the current path of the DUT inevitably causes a voltage drop across it. To be of any use, the voltage drop caused by the sensing element should be either negligible (even for large currents) or be made invisible to the DUT. The DUT will always see a well/stabilised supply voltage. Ensuring that the voltage drop is negligible, even for large currents, implies that a low ohmic sensing element should be used. This implies also that the voltage drop caused by small currents will also be very small. To measure this very accurate low voltage measuring techniques are required. Application of such techniques may either not be feasible due to a hostile signal environment or take too much time. The second drawback is that the sensing element also needs to pass the switching transients. To avoid a large voltage drop during the transients, the measuring device should be bypassed during the transients.

The advantage of the Keating-Meyer principle is that no special action needs to be taken in relation to transients, as the principle is relying on the parasitic capacitance of the DUTs supply pin and (if present) on the decoupling capacitance connected to the supply pin. The main drawback of the Keating-Meyer principle is that the supply of the DUT is left floating during the measurement phase. This would not cause any problem if it could be guaranteed that the voltage drop during the measurement phase is in all cases negligible. A negligible voltage drop once again requires very accurate very low voltage measurements. If the overcurrent is too high then it is possible that the supply voltage drops too much, leaving the circuit in an unknown state. Another drawback is that the measuring speed is function of the total capacitance.

6.3 QuiC-Mon v3.2

In this section a printed wiring bord called QuiC-Mon (Quiescent Current Monitor) version 3.2 is described. QuiC-Mon, developed by the university of New Mexico and Sandia Labs, reports an eight-bit value that may be converted into an absolute current measurement. It uses the Keating-Meyer method of I_{DDQ} measurement. This technique requires knowledge of the DUT V_{DD} pin capacitance (C_{DD}), see Figure 6-1; QuiC-Mon makes independent C_{DD} measurements up to 4 nF in the same vector as the I_{DDQ} measurement. The I_{DDQ} range is programmable, but typically up to 25 µA may be measured with the onboard Analog-to-Digital Converter (ADC) for a target I_{DDQ} resolution of 100 nA. I_{DDQ} values that fall outside the measurement range are signalled by a overrange output from the ADC. Range can be doubled through ADC cascading.

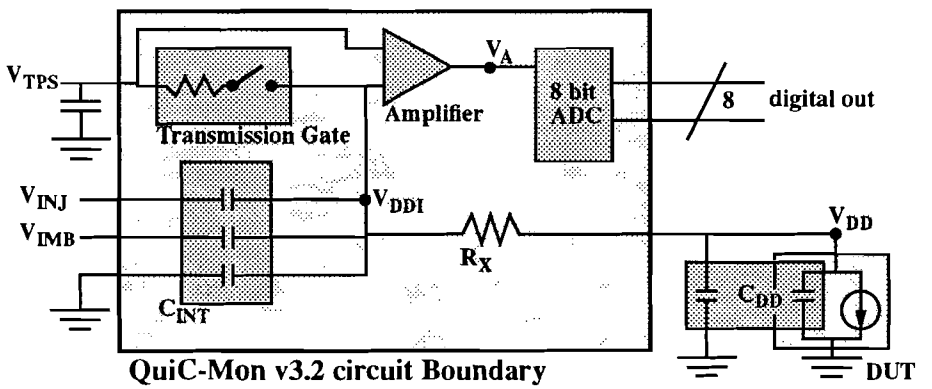


Figure 6-1: Block diagram of QuiC-Mon v3.2

Maximum test frequencies depend upon several factors, a primary one being the DUT power supply current (I_{DD}) transient settling time. Measurement rate also depends heavily on capacitance contributed by the DUT and load board. QuiC-Mon switches out most of the loadboard capacitance. Higher capacitance environments in the µF range slow I_{DD} measurements considerably, primarily because of RC time constants on the V_{DD} node introduced by QuiC-Mon. Despite these degradations, vector rates of over 250 kHz have been achieved on smaller VLSI devices with QuiC-Mon [Wallquist et-al, 93].

If the Transmission Gate switch is closed, the DUT draws current from the Tester Power Supply (V_{TPS} in Figure 6-1). If the Transmission Gate switch is opened, the DUT draws current from its own capacitance (C_{DD}). As a result, voltage on V_{DD} (and V_{DDI}) drops causing V_A to increase linearly. For typical I_{DDQ} measurements, the voltage drop in V_{DD} is on the order of 10 mV; in this range, it is assumed that even resistive defect loads will be linear. After a predefined discharge time has elapsed, the amplifier output is converted to a digital value representing measured I_{DDQ}.

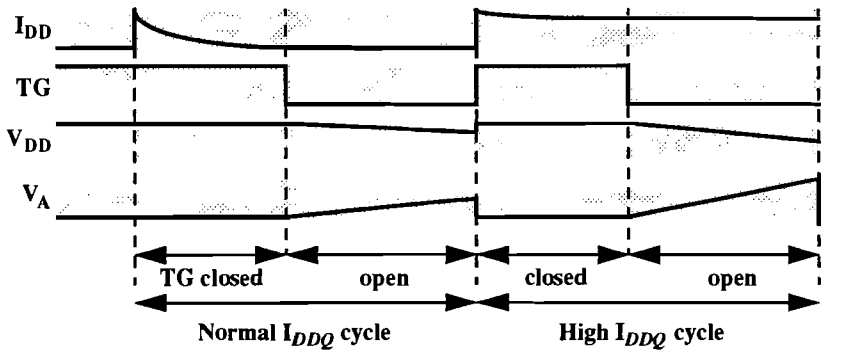


Figure 6-2: Typical Keating-Meyer Waveforms

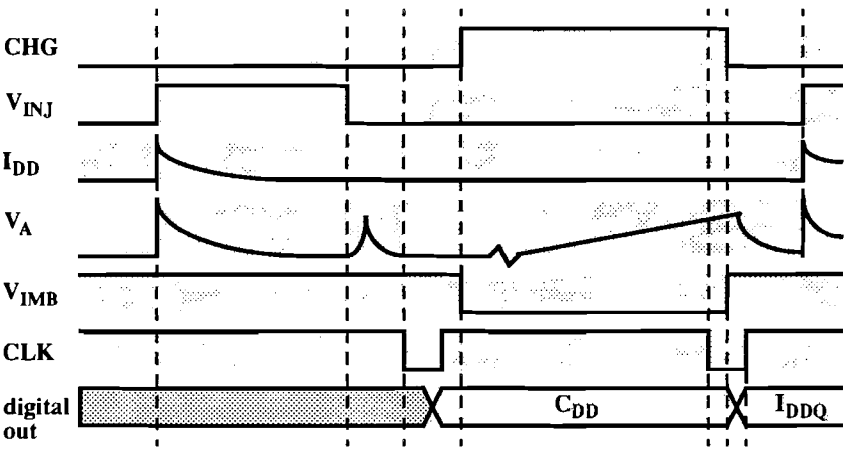


Figure 6-3: Typical QuiC-Mon v3.2 timing waveforms

- ***QTAG class & type:*** The QuiC-Mon v3.2 monitor clearly belongs to class 2 or 3, because of using a wiring board, limiting its physical size. It is a semi-digital measurement type monitor, because of its analog pins and external measurement equipment used for C_{DD} measurement.

- ***Monitor control:*** In literature describing QuiC-Mon v3.2 the typical Keating Meyer Waveforms (see Figure 6-2) are used to explain the measurement part [Wallquist et-al, 93]. In this figure the relation of I_{DD} and V_A is shown. In Figure 6-3 typical QuiC-Mon timing waveforms are shown. The CLK signal is used to latch the ADC output. The CHG input is a inverted version of TG described above. CHG is used in the final hardware version of QuiC-Mon. Note that the digital output is used for both I_{DDQ} measurement and C_{DD} measurement. Controlling this type of monitor fully automated is difficult. This is caused by the critical timing (floating V_{DD}) and continuous C_{DD} measurements needed for accurate I_{DDQ} measurements.

6.4 QuiC-Mon v5.0

This section decribes a new version of QuiC-Mon, further developed by one of the original authors with Philips Semiconductors. This monitor can be used for both I_{DDQ}/I_{SSQ}. Version 5.0 of QuiC-Mon takes the Keating-Meyer concept one step farther then version 3.2. The QuiC-Mon v3.2 allowed a simple calculation of the quiescent current (For a constant I_{DDQ}, the slope of the voltage drop at the node is constant) by measuring the amount of voltage drop after a predefined period of time, if C_{DD} was known. However, measuring current in this way has its drawbacks. The most obvious being a reliance on timing accuracy, as strobing at different times results in what can be considered different current gains. Longer wait times result in higher current resolution. Unless timing is guaranteed accurate, incorrect I_{DDQ} readings can result.

Instead of a direct voltage comparison, in Quic-Mon v5.0 a two stage measurement circuit is used. The first stage amplifier takes the derivate of the voltage at V_{SS}. This converts the constant slope waveform into a step function, producing a host of side benefits. The timing sensitivity is removed, and strobe placement is relaxed. Also, circuit settling time is improved significantly, allowing much faster measurement rates.

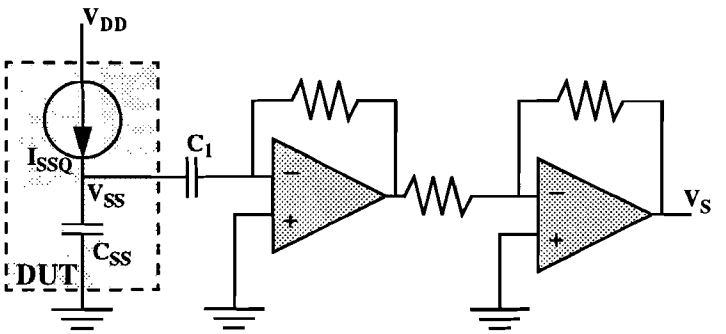


Figure 6-4: Model of QuiC-Mon v5.0 during I_{SSQ} measuring

One of the major drawbacks of QuiC-Mon v3.2 was the requirement of per-vector capacitance measurement. Version 5.0 overcomes this hurdle. A simplified model of QuiC-Mon v5.0 is shown in Figure 6-4. The variable gain, depending on the capacitance C_{SS}, can be avoided by making C_1 large with respect to C_{SS} [Wallquist, 94]. Quic-Mon's gain is now only dependent on the accuracy of the resistors, whose tolerance can be controlled to 1%. A side benefit is that small variations in DUT capacitance can now be tolerated.

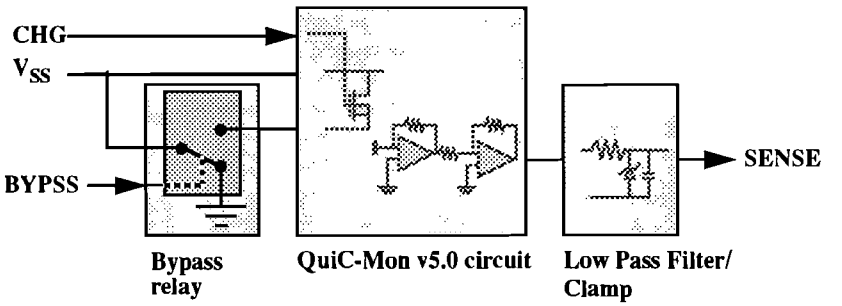


Figure 6-5: Block diagram of QuiC-Mon v5.0

In Figure 6-5 the block diagram of Quic-Mon v5.0 is shown. The bypass relay controls primary operation of the QuiC-Mon circuit. In its normally closed state, V_{SS} (the DUT V_{SS} pin) is shorted to ground. This allows full speed operation. When the relay is opened, QuiC-Mon becomes active. The Low Pass Filter/Clamp part is designed to protect the tester channel from excessive voltages and to reduce noise if I_{SSQ} is particularly noisy.

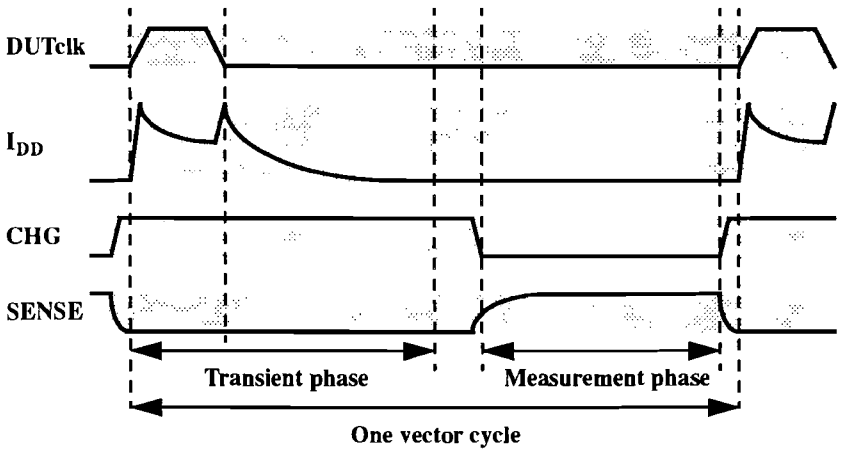


Figure 6-6: Typical QuiC-Mon v5.0 timing diagram

- ***QTAG class & type:*** QuiC-Mon v5.0 is build using a wiring board. This means it is a class 2 or 3 monitor, depending on its physical size. The analog output is used as pass/fail, so it is a semi-digital threshold monitor type.

- ***Monitor control:*** Timing on QuiC-Mon v5.0 is not as critical as in previous versions. In fact, setting up tests is actually quite simple. In Figure 6-6 a typical timing setup is demonstrated. One note should be made: Because the V_{SS} voltage is close to ground and may drop to a negative voltage with the charge injection onto V_{SS} after driving CHG low, the low drive voltage of CHG should be set below -2V.

  The transient phase (See Figure 6-6) extends from the first part input transition until

the power supply current settles. During this time, no measurements can be made. The length of this phase is entirely dependent on the DUT.

The measurement phase begins by driving CHG low and lasts until the amplifiers settle out. This can be as little as 700 ns (potentially longer if an output noise filter is used). Sampling of SENSE is done 50 ns before this phase ends.

6.5 OCIMU

The Off Chip Current Measurement Unit (OCIMU) is a monitor circuit developed by Alcatel-Bell and Technical School Ostende. The circuit differs from other available monitors in that it is capable of performing a relatively high speed (measurement time can be less than 30µs for a 2µF load) current measurement especially when driving a high capacitance load (up to several µF). The circuit is also able to deliver transient currents up to 10 amperes. Furthermore the circuit requires only a minimum of easily performed calibration and delivers a well-regulated supply voltage to the DUT never leaving the DUT supply pins floating.

The OCIMU circuit handles the problem of measuring I_{DDQ} in a hierarchical way. Before a detailed current measurement, the monitor circuit first decides if the actual I_{DDQ} current is within the 1µA - 1mA measuring range. Only when current is within the range, the measurement is activated. Otherwise an overcurrent indication is given, allowing the testing procedure to be speeded up. The OCIMU delivers a pass/fail output. The used threshold level can be set by a voltage level. For diagnostic reasons it has also an analogue output which is directly proportional to the I_{DDQ} current drawn by the DUT. A strobe signal delivered by the tester is used to trigger the monitor circuit to bypass the transient switching currents during DUT state changes.

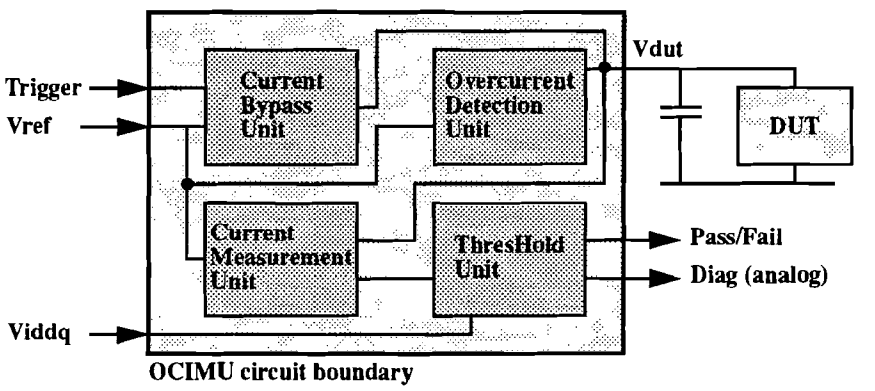


Figure 6-7: Basic building blocks of OCIMU circuit

The OCIMU circuit contains four basic building blocks, see Figure 6-7. These are a Current Bypass Unit (CBU), a Overcurrent Detection Unit (ODU), a Current Measurement Unit (CMU) and a ThresHold Unit (THU). The circuit has three inputs : Vref, Trigger and Viddq, and three outputs: Vdut, Pass/Fail and an analog diagnostic output (Diag). Vref is the refer-

ence for the regulating circuit, dictating the actual value of Vdut. Trigger is a control signal, supplied by the tester, which is used to activate a measurement cycle or to keep the monitor in bypass mode. Viddq is the programmable voltage reference for the pass/fail I_{DDQ} comparison. Vdut is the stable DUT supply voltage, Pass/Fail is a digital output indicating if the device passes or fails.

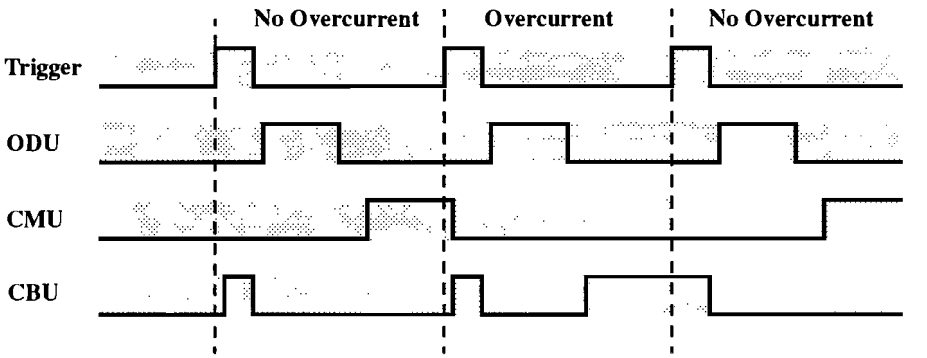


Figure 6-8: OCIMU circuit operation diagram

The OCIMU circuit operation is illustrated in Figure 6-8 and can be summarised as follows: The CBU is first activated by the high state of the externally provided Trigger signal. When the trigger signal is deactivated (changing to low) the CBU switches out in favour of the ODU which is active for a predefined minimum time, dictated by a monostable circuit. The ODU then performs a first measurement of the I_{DDQ} of the DUT and determines the order of magnitude. When the measured current is below the 1mA level then the ODU is switched out and the CMU is switched in to perform an accurate measurement in the 1µA - 1mA range and a comparison is made with a programmed reference value (Viddq) to generate a pass/fail output. If the current is above the 1mA level the ODU stays on and forces the Pass/Fail output into the fail state.

- ***QTAG class & type:*** OCIMU is a class 2 QTAG monitor that offers a digital pass/fail output but also uses analog pins (Vref, Viddq). This type of monitor is called a semi-digital threshold monitor.

- ***Monitor control:*** Controlling this type of monitor is easy. The signalling between the internal blocks is done without external signals. When Vref an Viddq are set, only a trigger signal as shown in Figure 6-8 has to be applied. This starts I_{DDQ} measurement and pass/fail information can be read after a predefined amount of time.

6.6 IDUNA-2

The IDUNA-2 monitor is a demonstration QTAG class 1 monitor developed by Philips Research and Lancaster University in ASIC form. It is a development from the original IDUNA-1 circuit, designed for on-chip I_{DDQ} monitoring, but adapted to match the requirements of the QTAG class 1 standard. The basic circuit has been improved to take advantage of both a modern CMOS process and better analog design methods.

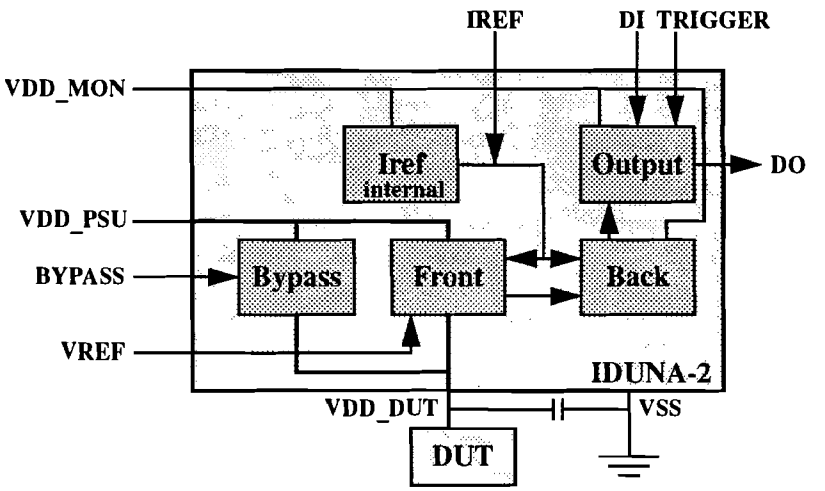


Figure 6-9: Top-Level block diagram of IDUNA-2

Functionality of the IDUNA-2 monitor is divided between a number of blocks as shown in Figure 6-9 and the interaction between these blocks changes according to what mode of operation the monitor is in. Two major modes are provided, the normal threshold based I_{DDQ} current pass/fail mode and the bypass mode.

If bypass mode is activated the Bypass block provides a short circuit resistance of typically two Ohms between VDD_PSU and VDD_DUT. Measurement of the current passing through the VDD_DUT pin by IDUNA-2 monitor is not possible in this mode although a pass/fail signal may still be generated which is not meaningful. Voltage regulation of the VDD_DUT pin is lost during bypass mode, the low resistance short means that the DC voltage of VDD_DUT will be close to that of VDD_PSU. Care must be taken when leaving bypass mode to set the VDD_DUT pin to the VREF voltage level. The lack of a large pull-down current internal to the IDUNA-2 monitor in addition to the large capacitance at the VDD_DUT pin means that un-assisted recovery time may be of the order of 1mS or more.

In measure mode current passing through the VDD_DUT input pin is passed through the Front block. The primary operation of this block is to mirror the current and passing the resulting current to a subsequent block of the monitor, the Back block. The current passed from the Front to Back block is compared with a current reference derived from an internal current reference cell. An asynchronous pass/fail decision regarding the current level of I_{DDQ} current

input is provided by a high impedance node that compares both currents. Adding or subtracting current from the IREF pin will alter the current threshold of the monitor. This is permissible as long as stated current limits are not exceeded.

The Front circuit is also used to regulate the voltage of the VDD_DUT pin to the externally set value of VREF irrespective of the magnitude of the current drawn through the VDD_DUT pin by the device. In this fashion, IDUNA overcomes one of the objections to the Keating-Meyer method, because the device's V_{DD} node is continuously driven. A potential problem with Keating-Meyer is that the V_{DD} pin is isolated from the PSU source.

Once the asynchronous pass/fail signal has been generated it is read by the Output block and may be latched to give synchronous pass/fail decisions using the Trigger input. The pass/fail signal may also be ANDED with previous pass/fail signals from other monitors using the DI input. This feature allows a daisy-chain of monitors to be created with a global pass/fail signal

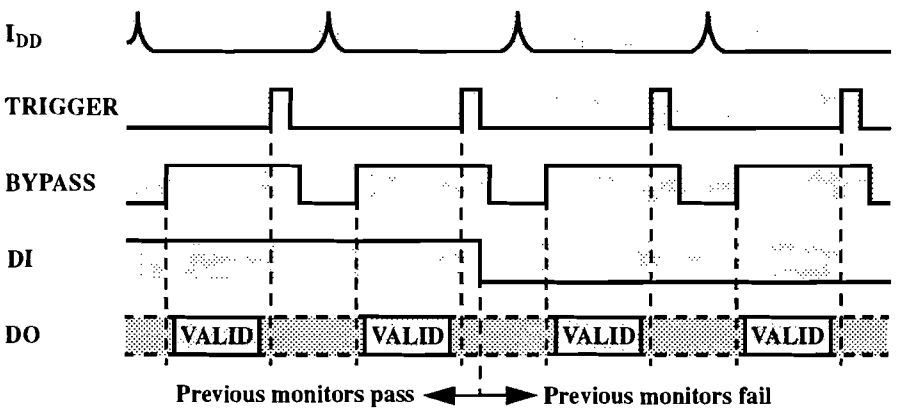


Figure 6-10: Typical IDUNA-2 synchronous timing diagram

- ***QTAG class & type:*** IDUNA-2 is a class 1 compliant I_{DDQ} monitor and is a semi-digital threshold monitor because both IREF and VREF signals must be set using either analog voltages or currents.

- ***Monitor control:*** The interface of IDUNA-2 to the test system follows closely the provisional QTAG standard interface of which pins are declared in section 5.5. The functions MODE, DI and DO are the same for all class 1 monitors [Hales, 94a], but for this monitor the MODE control signal, is in fact an explicit BYPASS control signal. More importantly, the IDUNA-2 monitor has an optional pin IREF that can be used to program the current threshold externally. However, if this pin is not connected the monitor uses the fixed internal current source. Similarly, the pass/fail flag data can be clocked by the TRIGGER pin where it is latched in a flip-flop. Figure 6-10 shows a typical IDUNA-2 timing diagram.

6.7 LTX I_{DDQ} monitor

The LTX I_{DDQ} monitor is a stand alone option of a LTX test system. It has been designed to integrate into the current test systems. The LTX trillium test head is modified to accept the I_{DDQ} monitor. The unit will be made with precision components that will not require calibration [LTX, 94].

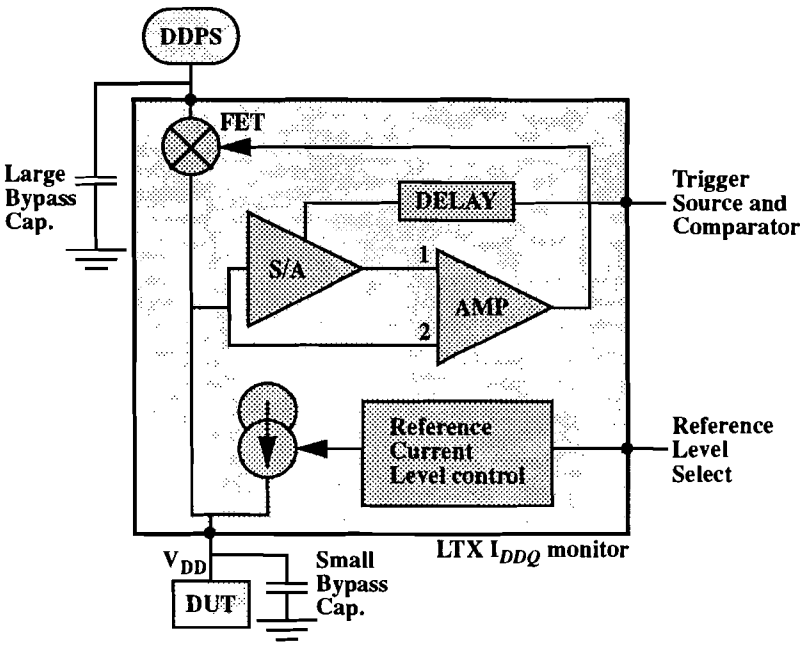


Figure 6-11: LTX I_{DDQ} monitor block diagram

Through the load board, the LTX I_{DDQ} monitor is connected to a standard DDPS (power supply). Non monitored V_{DD} pins bypass the monitor and monitored V_{DD} pins pass through the monitor. Figure 6-11 shows a basic functional block diagram of the I_{DDQ} monitor being used. Large decoupling capacitors (1µf - 100µf) are placed before the monitor. If any small decoupling capacitors (0.1µf - 0.01µf) are needed, they are applied after the monitor as close to the DUT as possible. Capacitors placed before the I_{DDQ} monitor will not affect the circuit performance.

Under normal vector operation the monitors FET is closed and I_{DD} is supplied from the DDPS.

On an I_{DDQ} vector an arming signal (See Figure 6-13) on the 'Trigger Source and Comparator' pin is applied from an unused tester channel. This arming signal opens the FET and after a delay, arms the sample and hold. This locks the V_{DD} voltage on input 1 of the operational amplifier. The I_{DDQ} pass/fail decision depends on two currents:

- The reference current
- The DUT I_{DDQ}

If the reference current is greater than the I_{DDQ} current, V_{DD} will increase due to charging of the DUT capacitance (including small bypass capacitors), see Figure 6-12(A). As a result, the voltage on input 2 of the operational amplifier (Figure 6-11) will increase producing a logic high. If the reference current is less than the I_{DDQ} current,V_{DD} will decrease due to discharging of the DUT capacitance (including small bypass capacitors), see Figure 6-12(B). As a result, the voltage on input 2 of the operational amplifier will decrease producing a logic low. Through a levels translator, the arming tester channel also accepts the logic level from the operational amplifier and latches the pass or fail into the tester. The arming signal is then unasserted, the FET closes, the DUTs I_{DD} is again supplied by the DDPS and the next vector is applied.

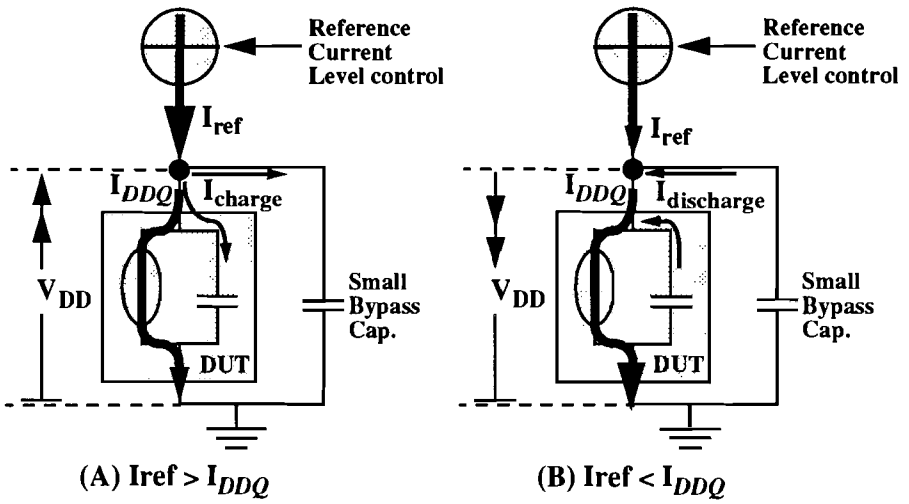


Figure 6-12: Model of currents with quiescent DUT

During I_{DDQ} testing the tester executes patterns at normal functional rates, on vectors which I_{DDQ} tests are performed the vector period is extended. The total cycle time needed for a I_{DDQ} vector will depend on the amount of capacitance on the DUT side of the monitor and the accuracy that is needed.

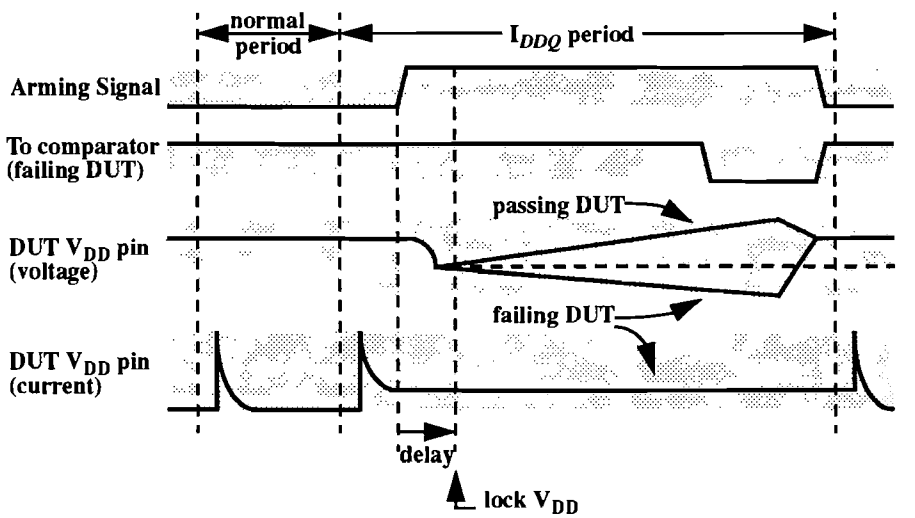


Figure 6-13: LTX I_{DDQ} timing diagram

- *QTAG class & type:* The LTX monitor is a monitor that is mounted in the test head of the tester. This is not a QTAG standard monitor, but it is controlled in the same way. This means MDF also can cover this type of monitor. According the QTAG standard it is a semi-digital threshold monitor in class 2 or 3.

- *Monitor control:* Controlling this type of monitor is done by digital pins connected to unused tester channels. In Figure 6-13 a timing diagram of the LTX monitor is shown. Threshold can be set using 4 pins, also connected to unused tester channels. The analog pass/fail output should be measured (compared) at predefined times.

6.8 The Fully Digital Interface[1]

The monitors described so far use an analog parameter to set the current threshold value. Some testers have a limited number of analog voltage sources, and they may not have any analog voltage sources available to set the threshold. This is even more likely when devices with multiple power supplies are tested because a separate I_{DDQ} or I_{SSQ} monitor, with its associated analog threshold signal, would be required for each power supply that is to be measured. This is an unrealistic requirement for some of today's production test environments. A fully digital interface [Hales, 94b] was proposed to the QTAG group as a way to make the QTAG monitors more useful in these environments.

The fully digital interface specifies the current threshold using a digital value which is loaded serially into the monitor rather than using an analog value. In addition, the serial interface enables multiple monitors to be chained together so that a single set of four digital tester pins can be used to control as many monitors as are required. Although this serial interface makes access to the monitors more complicated, it also provides more flexibility. It is thought that this interface can provide everything that is needed to control the complete range of QTAG monitors which will be available, ranging from the simplest low capability monitors to the more advanced monitors which will have current measurement capabilities as well as pass/fail current threshold checking.

6.8.1 A Simple Monitor

Figure 6-14 is a block diagram of the serial interface logic which would be used by one of the simpler QTAG I_{DDQ} monitors. The power connections at the bottom of the diagram (VDD_MON, VDD_PSU, VSS, and VDD_DUT) have the same functions as in the IDUNA-2 monitor. The other four connections are used to create the fully digital serial interface. The MODE, CLK, and DI pins are inputs to the monitor from the tester and the DO pin is an output from the monitor back to the tester. Multiple monitors can be connected by chaining together their DI and DO pins. The four pins which are used to create the serial interface are the same for both I_{DDQ} and I_{SSQ} monitors. The serial interface logic can be implemented in CMOS using less than 300 transistors together with a digital to analog convertor.

1. © *Copyright IEEE 1995*

MODE. This pin is used to switch between the monitor's *control* and *monitor* modes. *Control* mode is used to configure the monitor, and *monitor* mode is used to measure quiescent current values.

CLK. In *control* mode, the rising edge of **CLK** clocks serial data in and out of the monitor. In *monitor* mode, the rising edge of **CLK** indicates when a current measurement should be made.

DI. In *control* mode, this pin is used to load serial data into the monitor. In *monitor* mode, this pin is used to read the measurement result which is present at the **DO** output pin of the previous monitor in the chain.

DO. In *control* mode, this pin reads serial data from the monitor. When multiple monitors are chained together, this pin is also used to pass serial data from one monitor to the next monitor in the chain. In *monitor* mode, this pin is used to feed the measurement results back to the tester.

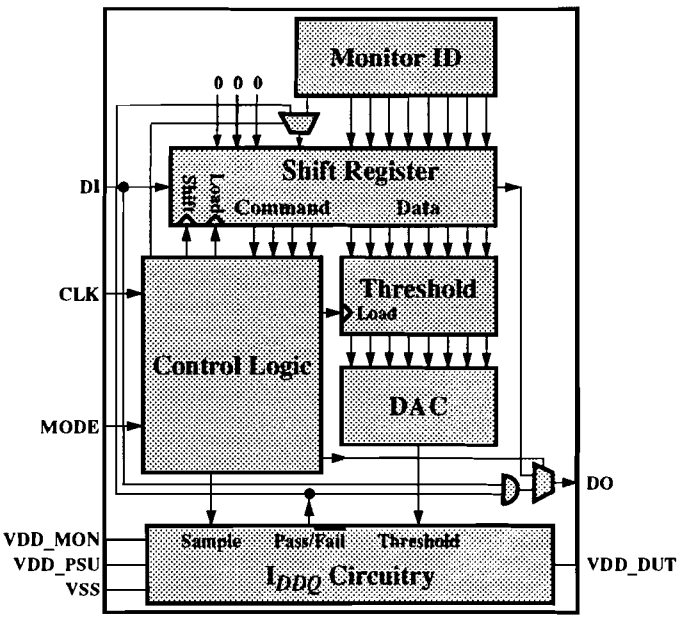


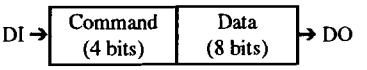
Figure 6-14: Block Diagram of Fully Digital Monitor

6.8.2 Control Mode

In *Control* mode the tester can set the monitor's threshold current, specify the current range, switch the monitor on and off, and perform various other control activities. This is done by loading serial data into the monitor via the **DI** input pin. When these data have been shifted through the monitor, they appear on the **DO** pin where they can then be loaded into the next monitor in the chain (if any). *Control* mode is also used to read serial data from the monitor back to the tester.

Serial access to the monitor is accomplished via a 12 bit internal shift register which is connected between the monitor's **DI** and **DO** pins. In *control* mode, data in this shift register are shifted by one bit on the rising edge of the **CLK** signal. The shift register is divided up into 4 command bits and 8

data bits. Data are shifted into the monitor starting with the least significant data bit, and ending with the most significant command bit. When the mode is changed from control mode to monitor mode, the command that is present in the 4 command bits of the shift register is executed. Most of the commands which have been defined are optional, and do not have to be implemented in all monitors.



Some of the commands cause the monitor to load various data into the monitor's internal shift register. These data can be read while the monitor is in *control* mode by shifting them out through the **DO** pin.

6.8.3 Monitor Mode

In *Monitor* mode the monitor measures the DUT's I_{DDQ}/I_{SSQ} value and sets the value of its **DO** pin to indicate the measurement result. When multiple monitors are used the monitors can be chained together to share a common set of four serial interface pins on the tester (Figure 6-15). When this is done, the tester only has direct access to a single monitor's **DO** pin, so the measurements results need to be combined to give a single pass/fail result. To do this, each monitor looks at the value that is present on its **DI** input pin as well as looking at its internal I_{DDQ}/I_{SSQ} measurement result. The monitor returns a $\overline{\text{fail}}$ condition if either its **DI** input pin or the result of its internal I_{DDQ}/I_{SSQ} measurement indicate a $\overline{\text{fail}}$ condition. When this is done, the **DO** pin of the last monitor in the chain acts as a combined pass/$\overline{\text{fail}}$ indicator for all of the monitors. In *monitor* mode, the tester should always apply a 1 (pass) to the **DI** pin of the first monitor in the chain.

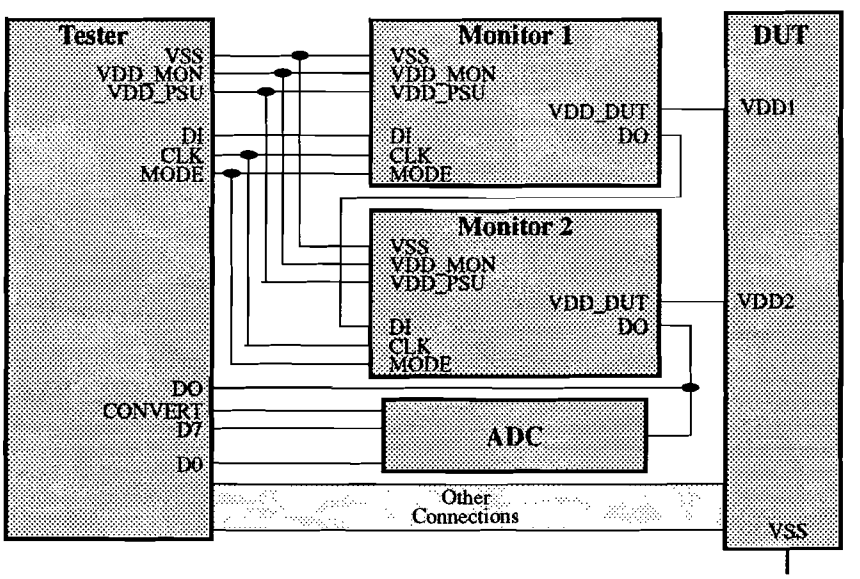


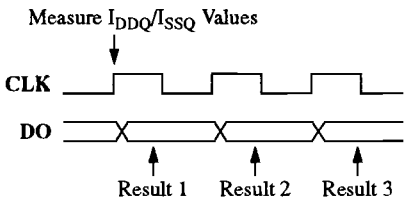
Figure 6-15: Block Diagram of Multi-Monitor Configuration

6.8.4 Current Measurements

The fully digital interface also provides two methods to support monitors which can return a current measurement rather than just returning a pass/fail result. The first method (the measurement monitor)

uses an integrated analog to digital convertor in the monitor, and the digitized result can then be accessed by shifting it out using the serial interface.

The second method (the analog monitor) is simpler to implement, but requires either an external analog to digital convertor or a mixed-signal tester with analog test capabilities. In this type of monitor the **DO** pin can also be used to output an analog voltage which is proportional to the sampled current value. Since multiple analog results cannot be combined into a single pass/fail value, the serial interface provides some commands which can be used to configure the monitors so that the analog measurement values appear sequentially at the output of the final monitor in the chain (shown below).



6.8.5 Connections to the Tester

Figure 6-15 shows how two fully digital I_{DDQ} monitors would be connected to the tester. The **DO** output of the first monitor is connected to the **DI** input of the second monitor. The pass/fail signal which appears at the **DO** pin of the second monitor will indicate a failure if the measurement from either of the monitors was above the specified threshold value.

Figure 6-15 also shows how *analog* I_{DDQ} monitors would be connected to the tester. The *analog* monitor's **DO** pin can return either a digital pass/fail indicator or an analog voltage. The pass/fail value is processed directly by the tester and the analog voltage is digitized by an external analog to digital convertor before it is passed on to the tester. In this case an eight bit analog to digital convertor has been used, so nine additional tester pins are required. The tester pin count could be reduced by adding a serial interface to the analog to digital convertor and connecting it using the existing serial interface bus. The pin count could also be reduced by feeding the analog voltage value directly to an input comparator of the tester where a pass-fail limit is set. Since this limit can be programmed from the tester software an external analog to digital convertor can be replaced by a slow, virtual convertor in the test program.

7 MDF definition

7.1 Introduction

As mentioned earlier in section 5.6 MDF is intended as a means of describing the key aspects of the implementation of I_{DDQ}/I_{SSQ} monitors used in conjunction with ATE systems [Baker, 94]. A similar approach as with BSDL has been chosen to come to the definition of MDF. The MDF description is based on the VHDL syntax, which means that the QTAG monitor is described in a VHDL entity description. In section 7.3 MDF is defined using Backus-Naur Form (BNF). The BNF conventions used are given in appendix A. The complete BNF description of MDF is given in appendix B. The MDF description starts with the definition of generic parameters. These parameters are needed to select default values used by test programs. These defaults include the package of the monitor, but also the format and state schemes used to control the monitor. The logical port description part defines the pin types of the monitor, the port names given are used as references in the rest of the MDF description. The monitor type part gives the possibilities of the monitor and also defines the minimum time period that can be used while testing. The package pin mappings are used to bind the port names to physical pins. The monitor control scheme is described in the same way as most tester software does, with format and state mappings. In MDF the state mappings are divided in two modes, bypass mode and monitor mode. These modes are needed to choose whether the monitor is active during a certain test or inactive (bypassed). The last part of the entity description consists of the monitor port identification. This part is used to bind the monitor pins to standard QTAG pin classes. These classes will first be discussed in the next section.

7.2 QTAG pin classes

QTAG monitors will be controlled by a combination of test vectors and test programs used to set voltage or current levels, measure values (analog or digital by down-loading captured data from pin memory), calibration and initialization [Baker, 94]. All test programs will be using a standard set of QTAG functions, available in all tester software belonging to QTAG compatible ATE. Each standard function will be dedicated to a predefined group of monitor pins. To group monitor pins, QTAG pin classes will be defined in this section. For a monitor pin, being member of a class implies the type of pin and its usage. MDF will contain information in the port identification part which binds all monitor pins to a QTAG class and gives parameter values to initialize the class-related standard functions. Below, the pin classes defined will be summarized.

- **QMON_BYPASS:**
  The bypass class is used for digital pins controlling the bypass mode of the monitor.

- **QMON_TRIGGER:**
  The trigger class is used for digital input pins latching output at specific time intervals.

- **QMON_DI:**
  This class is used for digital input pins which are used when several monitors are daisy chained together. It is an input for the DO signal from the previous monitor.

- **QMON_DO:**
  The DO class is used for digital output pins carrying pass/fail information. The pass/fail signal of the threshold monitor is present on the pins assigned to this class. It may be asynchronous or may be latched at specific time intervals using a pin assigned to the trigger class.

- **QMON_VDD_PSU:**
  Pins assigned to the VDD_PSU class are analog pins which are used to power the pins assigned to the VDD_DUT class. Current drawn through the VDD_DUT pins is taken from pins assigned to this class.

- **QMON_VDD_MON:**
  The VDD_MON class is used for analog pins which are used to power the monitor modules.

- **QMON_VDD_REF:**
  The analog pin class VDD_REF is used to define the voltage of the VDD_DUT class pins.

- **QMON_VDD_DUT:**
  Analog pins assigned to the VDD_DUT class are V_{DD} pins of the DUT.

- **QMON_IXXQ_REF:**
  The class IXXQ_REF is an analog pin class for pins that may be used to alter the fixed current threshold.

- **QMON_IXXQ_REF_BUS:**
  The class IXXQ_REF_BUS is used for digital pins that form a bus which is used to set the current threshold. The pins have to be declared as bit_vector in the logical port description.

- **QMON_VALUE_ANALOG:**
  The QMON_VALUE_ANALOG class is used for analog output pins (measurement type monitors) that have to be measured during a specific time in the monitor (measure) period.

- **QMON_VSS:**
  The V_{SS} class is used for pins that has the function of global ground.

7.3 The entity description

In VHDL all designs are expressed in terms of entities, which are the most basic building blocks in a design [Perry, 93]. MDF uses a VHDL entity description to describe current monitors with its pins, both analog and digital, its limitations and all information needed to control the monitor during quiescent current testing. For MDF, an entity has the following structure:

```
entity My_QTAG_mon is                          -- an entity for My_QTAG_mon
        <generic parameters>
        <logical port description>
        <standard use statement>
        {<use statement>}
        <monitor type>
        <device package pin mappings>
        <device port format mappings>
        <device default mode mappings>
        <monitor port identification>
end My_QTAG_mon;                               -- End description
```

This structure should be maintained with the order of elements as shown above to simplify non-VHDL applications. The elements will be addressed in the next sections.

7.3.1 Generic parameters

Generics are a general VHDL mechanism used to pass information to an instance of an entity. In MDF they are intended as a method for selecting among several packaging options that a device may have and also for selecting the default scheme (format, timing and states) used when using the QTAG monitor. The VHDL generic parameters must have the names shown in order for software to separate them from other parameters that might be passed to the entity. The construct has the following form:

```
generic ( PHYSICAL_PIN_MAP:    string:=<default package type>;
          PORT_FORMAT_MAP:     string:=<default format description>;
          BYPASS_MODE_MAP:     string:=<default bypass mode map>;
          MONITOR_MODE_MAP: string:=<default monitor mode map> );

          <default package type>       ::= "<VHDL identifier>"
          <default format description> ::= "<VHDL identifier>"
          <default bypass mode map>    ::= "<VHDL identifier>"
          <default monitor mode map>::= "<VHDL identifier>"
```

The VHDL identifiers refer to names given to constants with reserved MDF string types. The use of these parameters and the reserved string types will become clear in the next sections.

7.3.2 Logical port description

The device's system terminals are given meaningful symbolic names, which are used in subsequent descriptions. This allows the majority of the statements to be "terminal independent". The port description uses the VHDL port list in standard practice. All pins referenced in the MDF description must have been defined here, so not only digital but also analog pins involved with the current measurement must be defined. The port description form is:

port (<pin spec> {;<pin spec>});
<pin spec> ::= <identifier list>:<pin type><port dimension>
<identifier list> ::= <port name>{,<port name>}
<pin type> ::= **in** | **out** | **buffer** | **inout** | **linkage**
<port dimension> ::= **bit** | **bit_vector** (<range>)
<range> ::= <integer> **to** <integer> | <integer> **downto** <integer>

The <pin type> identifies the system usage of a pin, with the following options:

in	: simple input pin
out	: output pin which may participate in busses
inout	: bidirectional signal pin
buffer	: output pin that may not participate in busses
linkage	: analog pin

Note, every pin must have a unique name, so if there are several VSS pins for example, they must have different names such as VSS1, VSS2 etc, or be expressed as a vector. A vector refers to a port consisting of more than one pin. An example of a port statement is:

```
port ( VREF,VDD_MON,IREF,VDD_DUT,VDD_PSU:linkage bit;
       VSS:linkage bit_vector(1to2);
       DI,TRIGGER,BYPASS:in bit;
       DO:out bit );
```

This example port statement declares eleven pins, consisting of seven analog pins (five single linkage pins and one two pins linkage vector) and four digital pins (three input and one output).

7.3.3 Use statements

In VHDL commonly used data types are collected in a package. The MDF use statements identify a standard VHDL package and other optional user defined packages needed to define MDF attributes, types and constants. The following statement is mandatory in MDF:

```
use STD_QTAG_1_1995.all          -- Get standard MDF information
```

7.3.4 Monitor type

The type of QTAG monitor is defined by the use of VHDL attributes. An attribute contains data that is attached to a VHDL object. The used type definitions are declared in the standard QTAG VHDL package. Goal of the monitor type attributes is to add identification and verification capabilities to application software that has to control the monitor during testing. The following attributes describe the type of monitor:

attribute MONITOR_OPERATION **of** <component name> : **entity is** <monitor operation type>;
attribute MONITOR_CONTROL **of** <component name> : **entity is** <monitor control type>;
attribute MONITOR_RETURN **of** <component name> : **entity is** <monitor return type>;
attribute MONITOR_THRESHOLD **of** <component name> : **entity is** <monitor threshold type>;
attribute MONITOR_THRESHOLD_VALUE
of <component name> :**entity is** <real number>;
attribute MONITOR_THRESHOLD_LIMITS_RANGE
of <component name> :**entity is** (<limits range record>);
attribute MONITOR_MEASUREMENT_LIMITS_RANGE
of <component name> :**entity is** (<limits range record>);
attribute MONITOR_MIN_CLOCKPERIOD
of <component name> :**entity is** <real number>;

<monitor operation type> ::= **IDDQ** | **ISSQ** | **IXXQ**
<monitor control type> ::= **SEMI_DIGITAL** | **FULL_DIGITAL**
<monitor return type> ::= **MEASUREMENT** | **THRESHOLD** | **BOTH**
<monitor threshold type> ::= **FIXED** | **VARIABLE**
<limits range record> ::= <lower limit>,<upper limit>
<lower limit> ::= <real number>
<upper limit> ::= <real number>

The operation type IXXQ is used for monitors that can be used to monitor as well I_{DDQ} as I_{SSQ}. In all MDF descriptions operation, control, return and minimum clock period attributes are used. The other monitor type attributes are used depending on the facilities the monitor offers. For example when describing a monitor with fixed threshold only attributes MONITOR_THRESHOLD (is FIXED) and MONITOR_TRESHOLD_VALUE are used. The formal MDF rules can be found in appendix B.

7.3.5 Device package pin mappings

In this part of the description the VHDL attribute and constant statements are used to list the package pin mapping. The format is shown by example:

attribute PIN_MAP of** My_QTAG_mon : **entity is PHYSICAL_PIN_MAP ;

***constant** My_QTAG_mon_SOL16 : **PIN_MAP_STRING** := <pin map string> ;*

Attribute PIN_MAP is a string that is set to the value of the parameter PHYSICAL_PIN_MAP, as named above. VHDL constants are then written, one for each packaging version that describes the mapping between the logical and physical device pins. Note that the type of the constant must be PIN_MAP_STRING. An example of a pin mapping string is:

"VREF:2,DO:1,DI:15,VSS:(14,16),VDD_MON:13," &
"TRIGGER:11,IREF:9,BYPASS:7,VDD_DUT:5,VDD_PSU:3"

Note that the string is (arbitrarily) divided into two parts by the concatenation character "&", which has no further syntactical meaning. An MDF parser will read the concatenated content of the string and match portnames with the names in the port definition. The symbol on the right of the colon is the physical pin associated with that port signal. If signals like VSS are 'bit_vector' in the port definition, then a matching list of pins enclosed in parenthesis are required. The physical pin mapped onto VSS(2) is pin 16 in the above example.

7.3.6 Device port format mappings

The VHDL attribute and constant statements are used in a similar way to the device package pin mappings. However, here they are used to define different format schemes for the ports of the QTAG device. The format mapping form is:

attribute FORMAT_MAP of <component name> **: entity is PORT_FORMAT_MAP ;**

constant <format mapping name> **: FORMAT_MAP_STRING :=** <format map string> **;**

<format mapping name> ::= <VHDL identifier>
<format map string> ::= "<port format map>{,<port format map>}"
<port format map> ::= <port name>:<port format>
<port format> ::= (<format type>[,<leading edge>][,<trailing edge>])
<format type> ::= <digital format type> | <analog strobe type>
<digital format type> ::= **DA | RZ | RO | RC | NR | SB | DC | TR**
<analog strobe type> ::= **MV**
<leading edge> ::= <integer>
<trailing edge> ::= <integer>

Attribute FORMAT_MAP is a string that is set to the value of the parameter PORT_FORMAT_MAP, as named above. VHDL constants are then written, one for each format scheme used for different monitor usage. For example can be thought of a synchronous and an asynchronous format scheme. When no interactive application software is used, the used format scheme can be chosen through the generic parameter PORT_FORMAT_MAP. An example of a format mapping string is:

"DO:(SB,50,80),DI:(NR,0),TRIGGER:(NR,0),BYPASS:(RZ,10,90)";

The <format type> identifies the format of a pin, with the following options:

DA : Data (state changes only occur at the beginning and end of a period)
RZ : Return to Zero
RO : Return to One
RC : Return to complement
NR : No Return
SB : StroBe
DC : Dont Care
TR : TRistate

MV : Measure Value (strobe for analog pin)

The MV format is a special format that is defined to indicate when a analog signal is valid to be measured. It uses a leading and trailing edge to define the 'signal valid' window. This format can only be used with pins that belong to the QMON_VALUE_ANALOG class, mentioned in the port identification part.

Leading and trailing edges are given in percentages of the defined period length. The minimum period length (MONITOR_MIN_CLOCKPERIOD) is defined in the monitor type section of the MDF description.

7.3.7 Device default mode mappings

The VHDL attribute and constant statements in this part are also used in a similar way to the device package pin mappings, they are used to define different pin state schemes for as well analog as digital pins of the QTAG device. The state mappings are divided in two types. The first type is the state mapping of the pins used in bypass mode. In this mode, the QTAG monitor is not active. The second type is the state mapping of the pins used in monitor mode. In monitor mode the QTAG monitor is active. The mode mapping form is:

attribute BYPASS_MODE **of** <component name> : **entity is BYPASS_MODE_MAP;**
attribute MONITOR_MODE **of** <component name> : **entity is MONITOR_MODE_MAP;**

constant <mode mapping name>: **BYPASS_MODE_MAP_STRING:=** <mode map string>;
constant <mode mapping name>: **MONITOR_MODE_MAP_STRING:=** <mode map string>;

<mode mapping name>	::= <VHDL identifier>
<mode map string>	::= "<port mode map>{,<port mode map>}"
<port mode map>	::= <port ID>:<pin state>
<pin state>	::= <digital state>|<analog value>
<digital state>	::= **0 | 1 | L | H | X | Z**
<analog value>	::= <real number>

Attributes BYPASS_MODE and MONITOR_MODE are strings that are set to the value of the parameters BYPASS_MODE_MAP and MONITOR_MODE_MAP, as named above. VHDL constants are then written, one for each pin state scheme used for different tests. When no interactive application software is used, the used pin state scheme can be chosen through the generic parameters BYPASS_MODE_MAP and MONITOR_MODE_MAP. An example of a mode mapping string is:

"VREF:5.0,IREF:0.0,VDD_PSU:5.5,VSS(1):0.0,VSS(2):0.0," &
"DI:1,TRIGGER:1,BYPASS:1,DO:X"

The <digital state> identifies the state of a pin, with the following options:

0	: input pin low state
1	: input pin high state
L	: output pin low state
H	: output pin high state
X	: output pin don't care
Z	: output pin tristate

The value system follows that of IEEE Std_Logic_1164 for digital pins. The values given to the analog pins should be interpreted by the application software according to the monitor port identification.

7.3.8 Monitor port identification

The monitor port identification part describes to which class of QTAG pins the monitor pins belong. This class information is used by application software to identify the monitor pins and to initialize the class-related functions on these pins. The port identification is defined by the use of VHDL attributes:

```
attribute QMON_BYPASS          of <port name> : signal is (<level record>);
attribute QMON_TRIGGER         of <port name> : signal is (<level record>);
attribute QMON_DI              of <port name> : signal is (<level record>);
attribute QMON_DO              of <port name> : signal is (<level record>);
attribute QMON_VDD_PSU         of <port name> : signal is "<vdd rec str>";
attribute QMON_VDD_MON         of <port name> : signal is "<vdd rec str>";
attribute QMON_VDD_REF         of <port name> : signal is "<vdd rec str>";
attribute QMON_VDD_DUT         of <port name> : signal is "(<dep rec str>)";
attribute QMON_IXXQ_REF        of <port name> : signal is (<ref record>);
attribute QMON_IXXQ_REF_BUS    of <port name> : signal is (<ref bus record>);
attribute QMON_VALUE_ANALOG    of <port name> : signal is "<val_ana rec str>";
attribute QMON_VSS             of <port name> : signal is <default value>;
```

The identification attributes are all optional, but every port name declared in the port description part has to be bound to a QTAG pin class.

All digital pin classes are records assigned to, defining voltage levels belonging to logic low and high levels, used to control the monitor.

```
<level record>          ::= <low level voltage>,<high level voltage>
<low level voltage>     ::= <real number>
<high level voltage>    ::= <real number>
```

The analog pins bound to VDD classes, except the VDD_DUT class, use a <vdd rec str> string to define its function parameters. The string can be devided in three parts, defining the minimum, maximum and default voltage levels of the pin it is assigned to. The default value is defined by a real number, while the minimum and maximum values also can be defined using a record describing a dependency upon another analog pin. This linear relation is described using an offset and a multiplier.

```
<vdd rec str>           ::= <min value depend>,<max value depend>,<default value>
<min value depend>      ::= <real number> | (<dep rec str>)
<max value depend>      ::= <real number> | (<dep rec str>)
<default value>         ::= <real number>
<dep rec str>           ::= <port name>,<relation>
<relation>              ::= <offset>,<multiplier>
<offset>                ::= <real number>
<multiplier>            ::= <real number>
```

The VDD_DUT pins can only be described in a dependency, because the voltage level of these pins is always dependent on other pins. This is done in the same form as described in the previous paragraph. The default value will be generated using this dependancy and the default value of the pin it is dependent upon.

```
<dep rec str>           ::= <port name>,<relation>
<relation>              ::= <offset>,<multiplier>
<offset>                ::= <real number>
<multiplier>            ::= <real number>
```


The IXXQ_REF class parameters are defined using records with five real number fields. The first three are used to initialize minimum, maximum and default current levels, the remaining two (offset and multiplier) are used to convert the pin current level into the altered current threshold used when monitoring.

```
<ref record>      ::=  <min value>,<max value>,<default value>,<conversion>
<min value>       ::=  <real number>
<max value>       ::=  <real number>
<default value>   ::=  <real number>
<conversion>      ::=  <offset>,<multiplier>
<offset>          ::=  <real number>
<multiplier>      ::=  <real number>
```

The IXXQ_REF_BUS class controls the internal current threshold. This is done by digital pins which form a bus that can be given a certain logical value. This value is used to set the current threshold. The attribute used for this class consist of a record with four number fields and a string. The first two number fields are real numbers that define the voltage levels belonging to the logic low and high value as is standard practice with digital pins. The next two integers define the bit_vector numbers that identify the most significant bit (MSB) and least significant bit (LSB) used in the lookup table defined by using a string.

```
<ref bus record>     ::= <level record>,<msb ID>,<lsb ID>,<ref table string>
<level record>       ::= <low level voltage>,<high level voltage>
<low level voltage>  ::= <real number>
<high level voltage> ::= <real number>
<msb ID>             ::= <integer>
<lsb ID>             ::= <integer>
<ref table string>   ::= "<table entry>{,<table entry>}"
<table entry>        ::= <bus pattern>:<ref value>
<bus pattern>        ::= <bit>{<bit>}
<bit>                ::= 0 | 1
<ref value>          ::= <real number>
```

The usage is shown by the following example:

```
attribute QMON_IXXQ_REF_BUS of ADR : signal is (0.0,5.0,3,0,   "0000:10.0E-6," &
                                                               "0001:11.0E-6," &
                                                               "0010:12.0E-6," &
                                                               "0100:13.0E-6," &
                                                               "1000:14.0E-6");
```

In this example, bit_vector ADR is a four pins wide bus. The logic low and high levels correspond with voltage levels 0.0V and 5.0V. The MSB and LSB used in the lookup table are assigned to portID 3 and to portID 0. This means that ADR(3) is MSB and ADR(0) is LSB. The physical pins of the portIDs are described in the device package pin mappings. The lookup table contains current threshold values assigned to each table entry.

As described in the previous chapter, measurement monitors with an analog output will also be part of the QTAG monitor standard. The outputs of these monitors will be bound to the VALUE_ANALOG class. The class parameters are described using a string. This string contains minimum and maximum values and a linear conversion part in which an offset and mul-

tiplier are given. The minimum and maximum values can be real numbers but also dependencies as described in the VDD classes part. A time window, indicating the analog value is valid, is defined in the format mapping part using the MV (measure value) format.

```
<val_ana rec str>      ::= <min value depend>,<max value depend>,<conversion>
<min value depend>     ::= <real number> | <depend record>
<max value depend>     ::= <real number> | <depend record>
<depend record>        ::= <port name>,<relation>
<relation>             ::= <offset>,<multiplier>
<offset>               ::= <real number>
<multiplier>           ::= <real number>
<conversion>           ::= <offset>,<multiplier>
<offset>               ::= <real number>
<multiplier>           ::= <real number>
```

The last class defined is the VSS class. This class is only given a default voltage level by a real number attribute. All pins bound to this class are used as global ground.

```
<default value>        ::= <real number>
```

At this time QTAG pin classes have been defined based upon presently known monitors and proposals for future development. All types of monitor pins that can be expected on future QTAG monitors are covered. However, extension of these classes is foreseen.

In appendix F, three examples of MDF are given describing IDUNA-2, QuiC-Mon v5.0a and the LTX_I_{DDQ} monitor.

8 Using MDF in Philips CAD-Test flow

8.1 Introduction

Philips CAD-Test flow is based on a set of tools developed by Philips Semiconductor and Philips ED&T (Electronic Design and Tools) to implement the macro-test methodology devised by Philips Research [Bouwman,92]. In the macro-test methodology the Divide and Conquer approach to design for testability is employed to break the testing of an IC into individual macros that have a specific test generation tool for which a known fault or defect coverage can be generated. Hence random logic would be partitioned away from RAMs, and different analogue blocks would be partitioned into macros. Using this strategy, the individual test sequences must be first merged with the necessary test protocol, to transport the test data to and from the macro, and second, to merge all the macro test vectors into one efficient unified program.

8.2 Computer Aided Test (CAT) system

The set of tools developed by Philips Semiconductors and Philips ED&T used for test generation are part of the Philips Computer Aided Test (CAT) system. Pattern generation is done by four different software tools, see Figure 8-1. For random logic the combinatorial pattern generator AMSAL is used. For RAM and/or ROM the algorithmic generator MemGen is used. If a functional test is desired, the functional pattern converter FCon can be used to convert simulation listing files into a test pattern set. The boundary scan pattern generator TimPat is used to generate vectors for testing boundary scan circuitry (used for testing PCB).

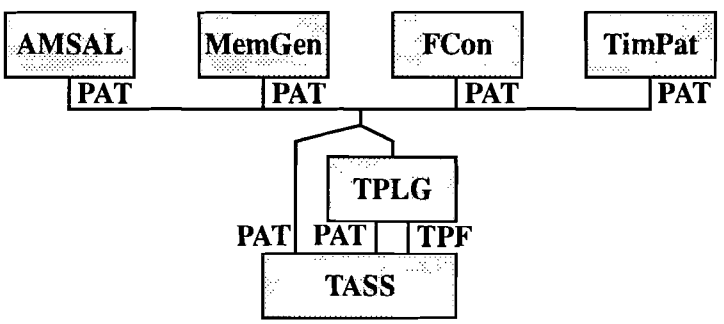


Figure 8-1: Pattern generation tools in Philips CAT system

All pattern generators produce a PAT file, the standard pattern file. This file contains stimuli and predicted responses, expressed with respect to pins or chains of the block to which it refers. It is the task of the Test ASSembler (TASS) to change this level of reference from block level to chip level and to resolve the chain references. How this needs to be done can be figured out by the Test PLan Generator (TPLG), which output is used to instruct TASS how to assemble tests. An important property of the PAT file is that is does not contain any timing-related information. It is up to TASS to add this information according to the user-specified needs.

8.3 TASS

TASS is probably a unique tool in the industry, because it is not only capable of merging test vectors and protocols of different macros into a unified test sequence for an IC. It can translate those vectors into a wide variety of different ATE vector formats and simulator formats, but also into VHDL and Verilog test benches. In Figure 8-2 a block diagram of TASS is shown. The input files of TASS consist of a control (CTR) file and test data files (TDF) consisting of pattern (PAT) files and Test Description Language (TDL) files.

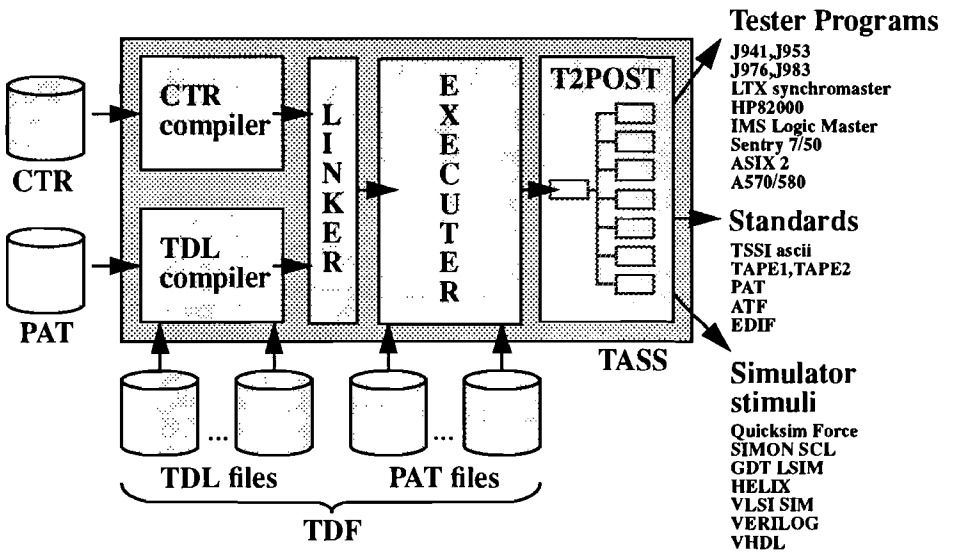


Figure 8-2: TASS block diagram

The TASS CTR file contains a control description in TASS Command Language (TCL). It is used to influence the execution of the TASS program and to specify the required input and output files. The timing information belonging to DUT test vectors is described in the same way as tester software does, using formats (waveforms) and edges combined in a waveset. The waveset information belonging to the test data files is also defined in the CTR file. The TDL files contain test descriptions that are used to manipulate test vectors.

As shown in the TASS block diagram (Figure 8-2) no data path is available from the CTR compiler to the TDL compiler. As a result, the test description language can not handle data that is defined in the CTR file. This means that the test description language can only change stimuli and expected data of the test. The accompanying waveset information defined in the CTR file is static during a test block. Even vector manipulations depending on waveset information are impossible. As will be shown later this is a major drawback when manipulating test vectors for I_{DDQ} test using a QTAG monitor.

The TASS output files are generated by the T2POST backend. This part contains the execution of the conditions which are defined in the CTR file and the calculation of the wavesets and delays. The output files can be divided in three groups. The first group consists of tester specific output files, the second group is formed by simulator stimuli and the third group is formed by standard files like PAT or Application Test Files (ATF). These files are used for

data transfer between different tools. The ATF format also can be used to execute multiple TASS runs on one vector set, see Figure 8-3. This can be necessary because of the poor concurrency TASS offers.

A great disadvantage of the multiple TASS runs using the ATF format is the lost of waveset information defined in previous runs because of the ATF files only containing stimuli and expected data without timing and waveform information.

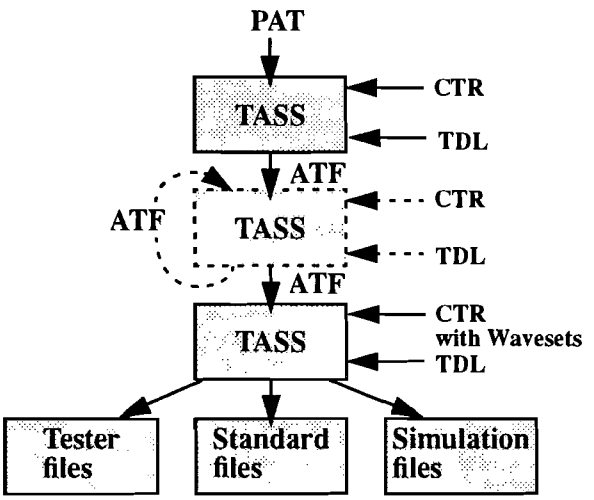


Figure 8-3: Multiple TASS runs using ATF

One facility that TASS cannot provide is test program generation. This is a disadvantage when using TASS in the QTAG Design-test flow (see Figure 5-3) because the monitor must to a degree be controlled via the test program. For instance, control of PSU voltages is only possible via the test program. Nevertheless for the purpose of creating a 'QTAG monitor APPEND' TASS is an ideal because it provides powerful vector manipulation facilities with the capability of direct translation to many ATE systems.

8.4 Philips I_{DDQ} Test-development flow

The QTAG standard has been defined to create a 'Plug&Play' environment for I_{DDQ} test. MDF also has been defined supporting this goal. The Philips I_{DDQ} Test-development flow also has to support the idea of 'Plug&Play' using a QTAG monitor. This means that it must be easy to append a QTAG monitor to existing test vectors. The ideal situation would be that appending a monitor to a complete test generated for a specific DUT does not need any regeneration of these DUT test vectors.

As mentioned in section 8.3 the 'QTAG monitor APPEND' part of the QTAG Design-Test flow can be implemented using TASS. It has the following two functions:

- Append monitor control to existing DUT test files
- Convert DUT testfiles keeping DUT quiescent during current measurement.

The digital monitor pins are connected to the tester pin electronics. This means they are controlled by the test vectors, which will be loaded into the tester pin memory. As a result, the monitor append consists of adding monitor pins and monitor control states to the DUT test vector files. The resulting test files will control both DUT and QTAG monitor. Keeping the DUT quiescent is not as easy as it seems using TASS. As mentioned before TDL, which is used to append the QTAG monitor to the DUT test vectors, does not offer the possibility of getting pin waveset information while processing vectors. Keeping the DUT in a quiescent state means that all DUT pins, including clock pins, may not switch during this period. To arrange this all DUT pins must be given a state, depending on the pin's format in the waveset description which is only available in the TASS CTR file. For example a pin with a Return to Zero format has to be held low (zero) to keep it quiescent.

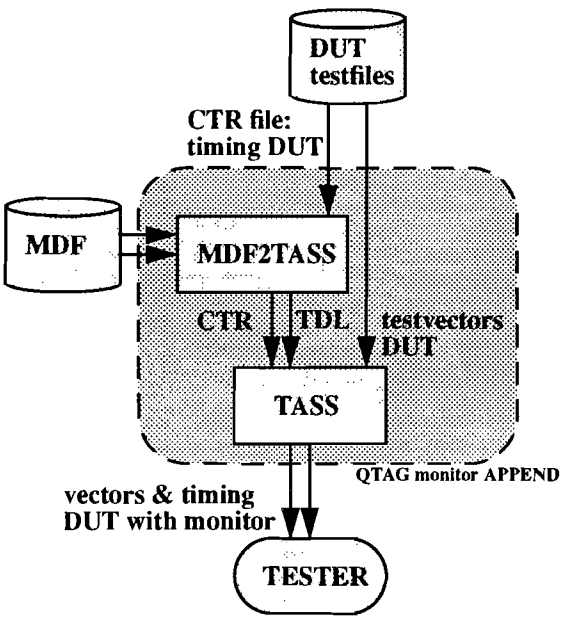


Figure 8-4: 'QTAG monitor APPEND' using TASS

The 'QTAG monitor APPEND' needed in the I_{DDQ} testflow using TASS has to contain a software tool that overcome the shortcomings of TASS in relation to the timing issue described. The resulting flow is shown in Figure 8-4. It has a 'QTAG monitor APPEND' that consists of a new developed MDF2TASS tool and a TASS-run on the DUT testvectors. MDF2TASS solves the problem described above by keeping a list with waveform information of all DUT pins (read from DUT CTR file). While generating the testvector manipulation functions in TDL this list is used to create functions, depending on the formats of the DUT pins. The TASS-run is controlled by a CTR file, containing timing information of the DUT and the QTAG monitor pins, and a TDL file, describing I_{DDQ} testvector manipulation functions, generated by MDF2TASS. The final result of the TASS-run are tester files for the DUT with a QTAG monitor on the test fixture.

The pattern files containing DUT test vectors consist of pin and chain declarations, input stimuli and expected data. These vectors describe long series of clock cycles in which data is scanned into scan chains. When all chains have been filled with stimuli, a 'normal' clock cycle occurs bringing the DUT in a state which is compared to expected data while shifting out the scan chains. I_{DDQ} is measured prior to a 'normal' clock cycle.

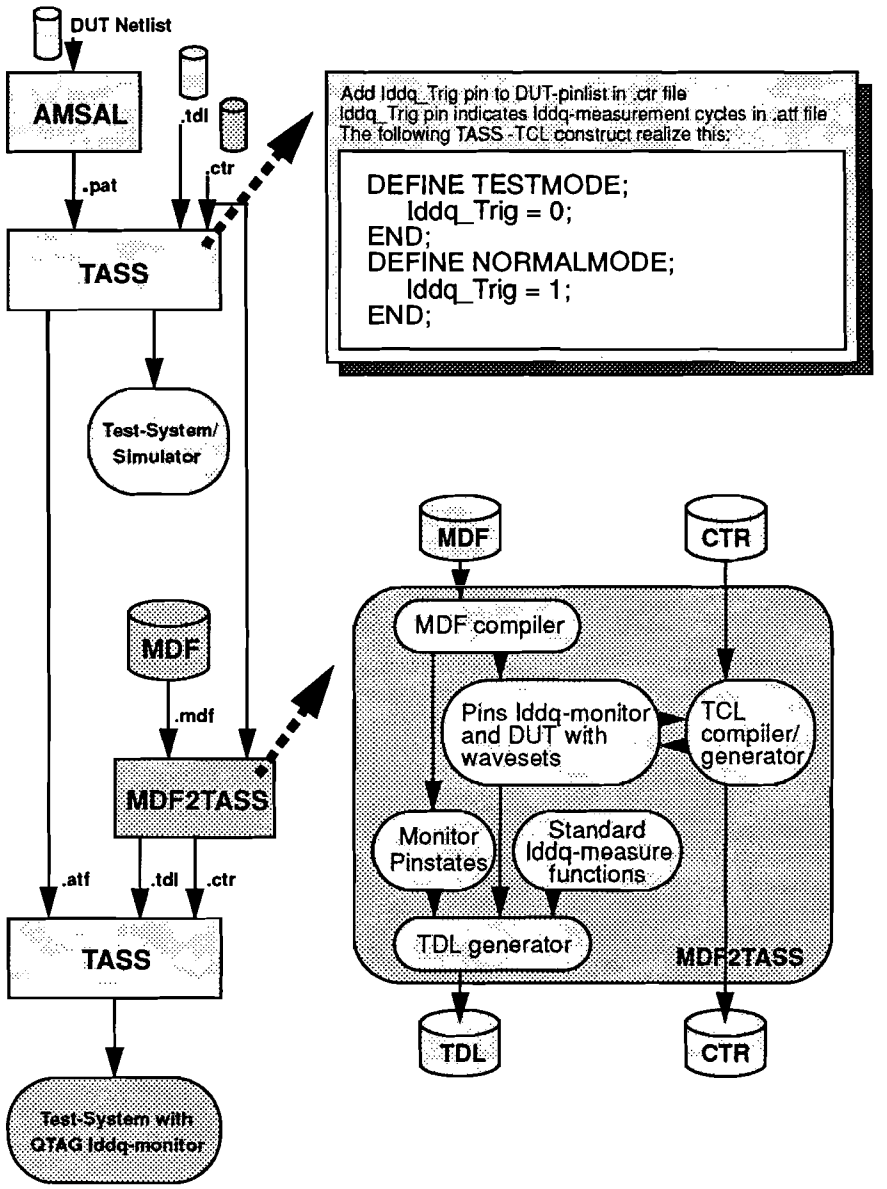


Figure 8-5: Philips I_{DDQ} Test-development flow

A 'QTAG monitor APPEND' needs a trigger flag in the DUT vector file indicating when I_{DDQ} has to be measured. When no flag is available the append tool cannot detect whether I_{DDQ} should be measured or not because the vector file does not always contain explicit information on the DUT being in scan mode or normal mode. For this reason precautions have to be taken assembling a test containing I_{DDQ} test vectors. After assembling such test in the output vector file a virtual pin must be available which will be used as I_{DDQ} trigger flag as

described above. Ideal would be using test pattern generators that generate this trigger information itself. This would allow combining I_{DDQ} test patterns and voltage test patterns in one test and this also would be better for a total 'Plug&Play' concept because adding a virtual pin during assembling the DUT test can be seen as a regeneration of existing DUT test vectors. This is not ideal as stated before. When adding trigger information during assembling special effort is asked from the test engineer who in the ideal 'Plug&Play' situation should not notice the difference of test generation with or without I_{DDQ} trigger flag.

In Figure 8-5 the Philips I_{DDQ} Test-development flow is shown. The pattern generator AMSAL is used for generating I_{DDQ} test patterns (without I_{DDQ} trigger flag). The creation of a virtual trigger pin called Iddq_Trig is done in the first TASS run used for assembling the DUT test without QTAG monitor. The TCL code used to generate this pin is also shown. The resulting DUT test vectors will contain a trigger pin which indicates I_{DDQ} measure cycles (Iddq_Trig = 1). In Figure 8-5 also the new developed tool MDF2TASS is shown, including its data flow which is magnified in Figure 8-6. In the next section this tool is discussed in detail.

8.5 MDF2TASS

A new tool called MDF2TASS has been developed to append QTAG monitor control to generated test vectors using the Philips CAT system. MDF2TASS has been written in ANSI C using YACC and LEX for building input file parsers.

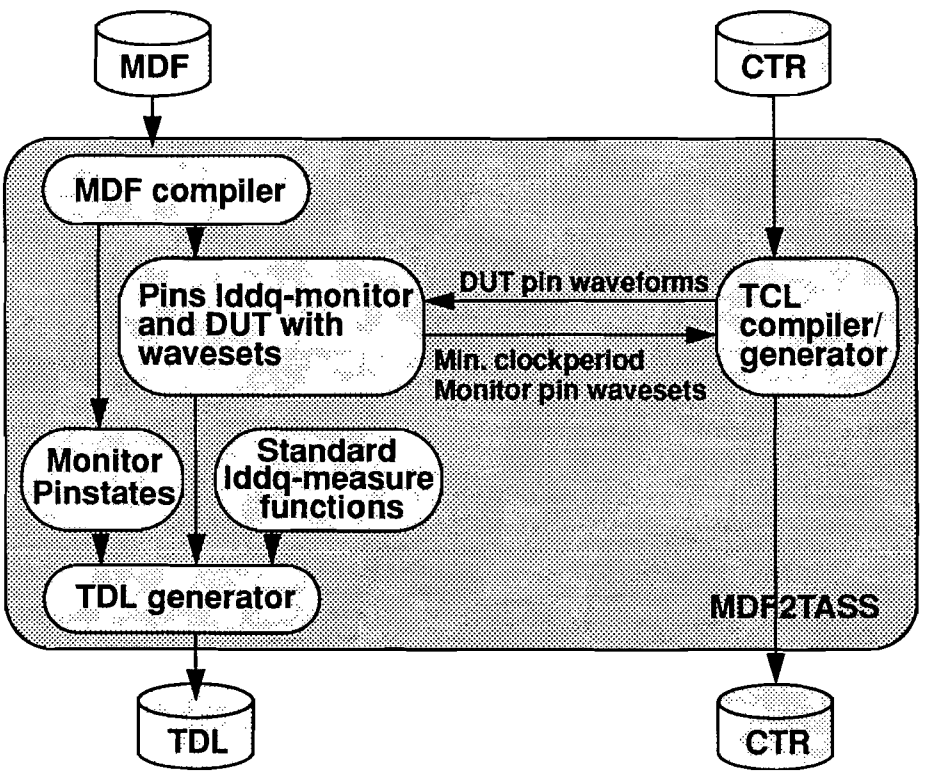


Figure 8-6: MDF2TASS data flow

In this section the functioning of MDF2TASS will be discussed on the basis of Figure 8-6. First, all tasks of MDF2TASS are summarized:

- Parse Monitor Description Format and build QTAG monitor data structure
- Parse TASS Command Language (CTR file) and build list of DUT-pin waveforms
- Generate CTR file with wavesets for DUT and QTAG monitor
- Generate I_{DDQ} test description in TDL, keeping DUT quiescent while measuring

When the MDF2TASS tool is executed, first the monitor MDF file is parsed by a MDF compiler. This compiler uses a YACC and LEX parser. The grammatical rules defined in YACC are a one to one translation of the MDF entity description in BNF (see appendix B). The lexical elements defined in LEX are a subset (same as BSDL) and standard practice of those of VHDL as defined in IEEE Std 1076. The most important elements are mentioned in appendix D. While parsing MDF a data structure is build containing all monitor specific information. The next step is formed by executing the TCL compiler/generator. The YACC and LEX files used for the TCL parser have been based on the grammatical and lexical rules defined in the TASS reference manual [CAT team, 94]. This compiler/generator behaves as a filter that copies relevant information directly from input to output CTR file. When DUT-pin timing and waveform information is read, it is also copied to the output CTR file and stored for later use while generating test functions in TDL. Only timing and waveform information defined in 'TCL global scope' is copied. While copying TCL from input to output CTR file pin declarations and wavesets are extended with QTAG monitor pins and accompanying wavesets read from the MDF data structure. The DUT test period, defined in the input CTR file, is overruled by the minimum test period defined in MDF attribute MONITOR_MIN_CLOCKPERIOD. When the TCL compiler/generator has finished, the TDL generator is executed. This part of the MDF2TASS software has access to the data structure containing QTAG monitor information read from the MDF file and to the DUT-pin waveform data. The generator contains standard I_{DDQ}-measure functions that only have to be completed with monitor and DUT specific pinnames and states. These functions are:

- monitor_bypass_mode: Generate monitor stimuli and expected data controlling monitor in bypass mode.
- monitor_measure_mode: Generate monitor stimuli and expected data controlling monitor in measure mode.
- get_quiescent_dut: Generate DUT stimuli keeping it quiescent.
- get_pattern_dut: Get original DUT test vector.

8.5.1 TDL flow generated by MDF2TASS

In TDL a main routine is written that calls the generated functions mentioned above while manipulating the DUT test vectors. Depending on the state of Iddq_Trig a function call sequence is executed, see Figure 8-7. The DUT test patterns are numbered. These numbers are used to reference DUT patterns while generating the new patterns including monitor control. The main routine starts by initializing a local pattern number counter and getting the first DUT test pattern referring to it. When the Iddq_Trig state is zero, the QTAG monitor will be kept in bypass mode. The accompanying monitor pin states are added to the DUT test pat-

terns by the function 'monitor_bypass_mode'. After this the output pattern is written and the local pattern number counter is increased. When the Iddq_Trig state is one, I_{DDQ} has to be measured before clocking the DUT. First the QTAG monitor control is added to the DUT test pattern by the function 'monitor_measure_mode'. Hereafter DUT-pin states are changed keeping the DUT quiescent during the current measurement cycle. These changes are made by the 'get_quiescent_dut' function.

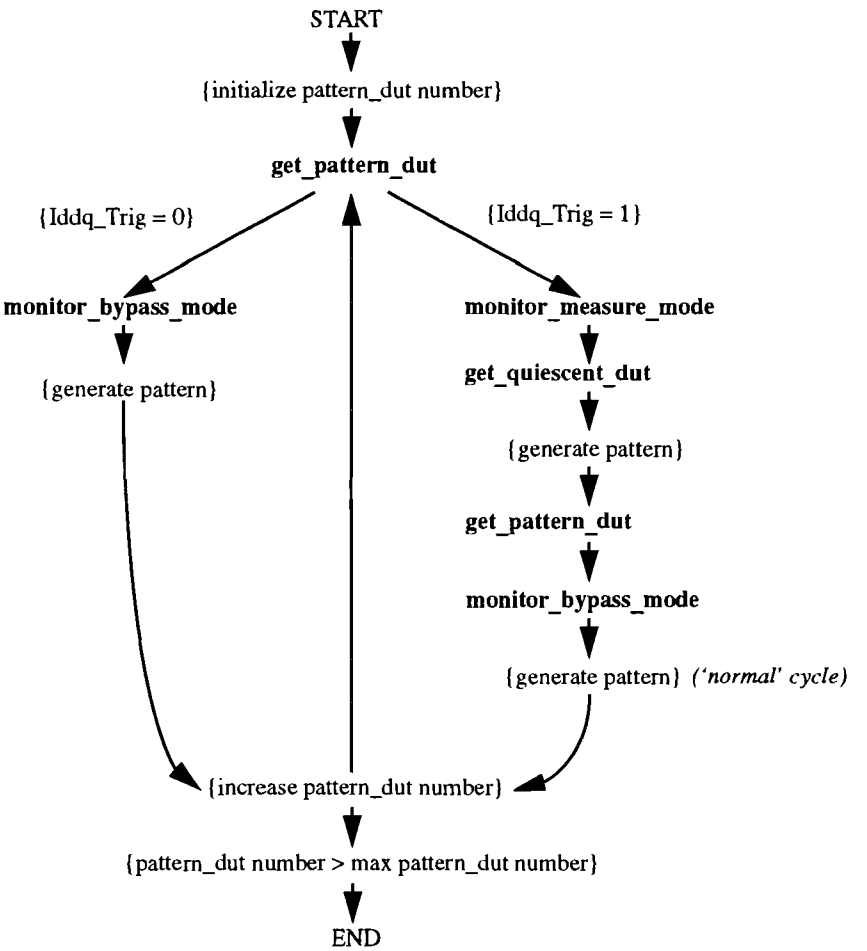


Figure 8-7: TDL flow generated by MDF2TASS

Now a new pattern is written containing the QTAG monitor control for measuring I_{DDQ} and the DUT-pin states keeping the DUT quiescent. The next step is formed by a reload of the original DUT pattern (without changes made to keep it quiescent) by the function 'get_pattern_dut' without increasing the local pattern number counter. The QTAG monitor bypass control is added calling the function 'monitor_bypass_mode'. The pattern written at this point contains the original 'normal' cycle. In the resulting test file all original 'normal' cycle vectors have been replaced by two vectors. One vector controlling a QTAG monitor for measuring I_{DDQ} with a quiescent DUT and one vector describing the usual 'normal' cycle keeping the QTAG monitor in bypass mode. In the next section an example is shown creating I_{DDQ} test vectors with MDF2TASS and TASS. The TDL file generated in this example has been printed in appendix E.5.

8.6 MDF2TASS example

In this section an example is shown in which MDF2TASS is used in the Philips CAD-Test flow to append Philips IDUNA-2 monitor control to simple test patterns, specially defined for the MDF2TASS example. In Figure 8-8 the I_{DDQ} Test-development flow and file names used for this example are shown. First, the pattern definitions in the pattern file 'testdut.pat' are explained, then the Application Test Files in the Test-development flow are discussed. The example is finished by vector generation for HP82000 tester.

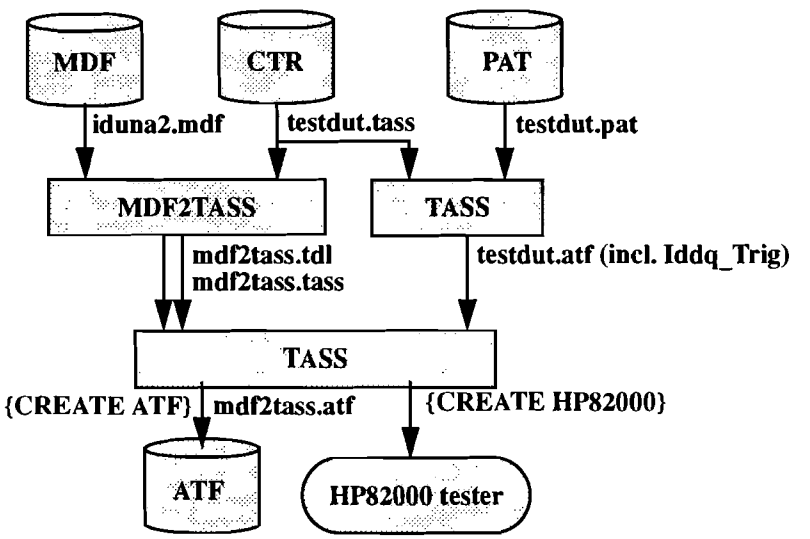


Figure 8-8: MDF2TASS example I_{DDQ} Test-development flow

All files, except 'iduna2.mdf', mentioned in Figure 8-8, have been printed in appendix E. In appendix F.1, the MDF description of Philips IDUNA-2 monitor ('iduna2.mdf') has been written.

8.6.1 Example test patterns

The MDF2TASS example uses a pattern file 'testdut.pat' which is generated by hand. This file contains two test pattern definitions which are discussed after explaining the test DUT they are defined for. In Figure 8-9 the test DUT is shown. It only consists of two scan chains. One scan chain (scani1 -> scano1) of length four and one scan chain (scani2 -> scano2) of length two.

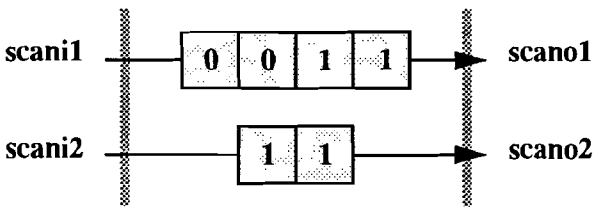


Figure 8-9: Test DUT used in MDF2TASS example

This information also can be found in the header definition of the pattern file. The header definition describes at which locations in a pattern inputs and outputs of the DUT can be found and at which positions the scan chain data has been written. Below a part of the pattern file 'testdut.pat' (see appendix E.1) is presented including additional comment.

Example: part of DUT pattern file: 'testdut.pat'

```
Header Definition;
      INPUT    scani1 1, scani2 2; -------->'normal' cycle input at position 1 and 2 in pattern
      OUTPUT   scano1 15,scano2 16; ------->'normal' cycle exp. data at position 15 and 16 in pat.
      CHAIN    scani1 3, scano1 9, 4, N; -->chain1: input start at 3, exp. data start at 9, length 4
      CHAIN    scani2 7, scano2 13, 2, N; ->chain2: input start at 7, exp. data start at 13, length 2
End;

Pattern Definition;
      1    : 01 1111 11 LLLL HH LL;
      2    : 10 0101 10 HHLH LH HH;
End;
```

The patterns defined in the above example describe 5 cycles each. These cycles consist of four scan cycles (longest scan chain length) and one 'normal' cycle. In Table 8-1 the extracted cycles of the two pattern definitions are shown. The chain input stimuli and expected data defined in the pattern definition have been completed with zeros for don't care input states and undefined (U) for don't care expected data.

Table 8-1: Cycles defined in 'testdut.pat'

cycle	scani1	scani2	scano1	scano2
scan 1	**1**	0	U	U
scan 2	**1**	0	U	U
scan 3	**1**	**1**	U	U
scan 4	**1**	**1**	U	U
normal	**0**	**1**	**L**	**L**
scan 1	**0**	0	**L**	**H**
scan 2	**1**	0	**L**	**H**
scan 3	**0**	**1**	**L**	U
scan 4	**1**	**0**	**L**	U
normal	**1**	**0**	**H**	**H**
scan 1	0	0	**H**	**L**
scan 2	0	0	**H**	**H**
scan 3	0	0	**L**	U
scan 4	0	0	**H**	U

Note that the expected data has been delayed. The expected data of pattern one is used together with the input stimuli of pattern two (While shifting new data in, expected data is compared to the data shifted out).

8.6.2 DUT Application Test File

As shown in Figure 8-8 an ATF file 'testdut.atf' (see appendix E.3) is generated by TASS controlled by the 'testdut.tass' (see appendix E.2) CTR file. While generating the DUT test vectors a I_{DDQ} trigger pin is defined. At this point for example HP82000 test vectors could be created (instead of ATF), used for testing the DUT without QTAG I_{DDQ} monitor. The Iddq_Trig pin in this case can be used to trigger I_{DDQ} measuring resources. During the MDF2TASS example at this point an ATF file is created. Below a part of this ATF file has been copied:

Example: part of ATF file: 'testdut.atf'

```
HEADER DEFINITION;
     INPUT     SCANI1 1, SCANI2 2, CLK 3, IDDQ_TRIG 4;
     OUTPUT    SCANO1 5, SCANO2 6;
END;

PATTERN DEFINITION;

*  ***** START OF 1.TESTMODE   FOR DTB TESTDUT *****
     1 : 1010UU;
     2 : 1010UU;
     3 : 1110UU;
     4 : 1110UU;
*  ***** START OF 1.NORMALMODE FOR DTB TESTDUT *****
     5 : 0111LL;
*  ***** START OF 2.TESTMODE   FOR DTB TESTDUT *****
     6 : 0010LH;
     7 : 1010LH;
     8 : 0110LU;
     9 : 1010LU;
*  ***** START OF 2.NORMALMODE FOR DTB TESTDUT *****
     10: 1011HH;
*  ***** START OF 3.TESTMODE   FOR DTB TESTDUT *****
     11: 0010HL;
     12: 0010HH;
     13: 0010LU;
     14: 0010HU;
END;
```

As can be seen in the ATF file, the original pattern definitions of the input PAT file have been extracted in the ATF file. For this reason the Iddq_Trig pin has to be added in the first TASS run used on original PAT files. After extracting the patterns, explicit information of the DUT being in normal mode or scan mode has been lost. Note that adding the I_{DDQ} trigger pin is only possible when all pattern definitions describe I_{DDQ} patterns. When combining columns 1,2,5,6 of the ATF pattern definition, Table 8-1 arises. Column 3 contains the clock state of the DUT. As can be seen the clock state is a continuous one. Together with the accompanying waveform definition in the 'testdut.tass' CTR file a continuous clock signal is produced on the tester (if tester files are generated). The Iddq_Trig pin state is zero in scan cycles and one in 'normal' cycles as defined in section 8.4

8.6.3 Adding IDUNA-2 monitor

Adding IDUNA-2 monitor control to the DUT test vectors (ATF file) is done by executing TASS controlled by MDF2TASS output files. When MDF2TASS is called without parameters the following message is displayed:

Screencopy MDF2TASS call:

```
>mdf2tass

usage: mdf2tass <MDF file><TCL file><ATF filename><CREATE options>

example: mdf2tass iduna2.mdf iddqtest.tass iddqtest.atf atf no_compress
```

The MDF2TASS tool needs the following input parameters:

- <MDF file> : filename of MDF file
- <TCL file> : filename of TCL file that was used for creating the DUT ATF file
- <ATF filename> : filename of DUT ATF file. (ATF file is opened by TASS)
- <CREATE options> : TCL CREATE options, used for defining TASS output format

Below two examples of calling MDF2TASS are given. Both are used in the MDF2TASS example I_{DDQ} Test-development flow shown in Figure 8-8. The first call is used to create an ATF file ('mdf2tass.atf', see appendix E.6) including IDUNA-2 monitor control. The second call is used to generate HP82000 tester files. These files have been used to get a graphic representation of the DUT test vectors with additional monitor control.

MDF2TASS calls used for example:

```
mdf2tass iduna2.mdf testdut.tass testdut.atf atf no_compress

mdf2tass iduna2.mdf testdut.tass testdut.atf hp82000
```

Executing TASS with the files generated by MDF2TASS is done as follows:

TASS call used after executing MDF2TASS:

```
tass mdf2tass
```

While executing TASS, controlled by the files generated by MDF2TASS, tester files or other format files are generated used for testing a DUT with QTAG monitor on the test fixture. Below the header and pattern definition part of the ATF file ('mdf2tass.atf') generated by the final TASS run of the MDF2TASS example has been printed. As can be seen, IDUNA-2 monitor pins have been added to the header definition. Also columns are added in the pattern definition containing monitor control states and expected data.

The most important issue to note is the existance of additional cycles in which I_{DDQ} is measured. During these cycles the clock state (column 7) is zero, keeping the DUT quiescent, the bypass pin state is one and DUT output is ignored.

Example: part of ATF file: 'mdf2tass.atf'

```
HEADER DEFINITION;
     INPUT      QM_DI 1, QM_TRIGGER 2, QM_BYPASS 3, SCANI1 5, SCANI2 6,
                CLK 7, IDDQ_TRIG 8;
     OUTPUT     QM_DO 4, SCANO1 9, SCANO2 10;
END;

PATTERN DEFINITION;
* ***** TESTMODE ******** *****
     1 : 100U1010UU;
     2 : 100U1010UU;
     3 : 100U1110UU;
     4 : 100U1110UU;
* ***** IDDQ MEASURE MODE *****
     5 : 111H0101UU;
* ***** NORMALMODE ****** *****
     6 : 100U0110LL;
* ***** TESTMODE ******** *****
     7 : 100U0010LH;
     8 : 100U1010LH;
     9 : 100U0110LU;
     10: 100U1010LU;
* ***** IDDQ MEASURE MODE *****
     11: 111H1001UU;
* ***** NORMALMODE ****** *****
     12: 100U1010HH;
* ***** TESTMODE ******** *****
     13: 100U0010HL;
     14: 100U0010HH;
     15: 100U0010LU;
     16: 100U0010HU;
END;
```

In Figure 8-10 a timing diagram of the generated HP82000 test is shown. In this figure the IDUNA-2 monitor control can be observed. When trigger IDDQ_T is high, I_{DDQ} has to be measured. DUT clock CLK is held low and bypass mode is left (QM_BYP high). After triggering the monitor (QM_TRI) the pass/fail is observed at output QM_DO.

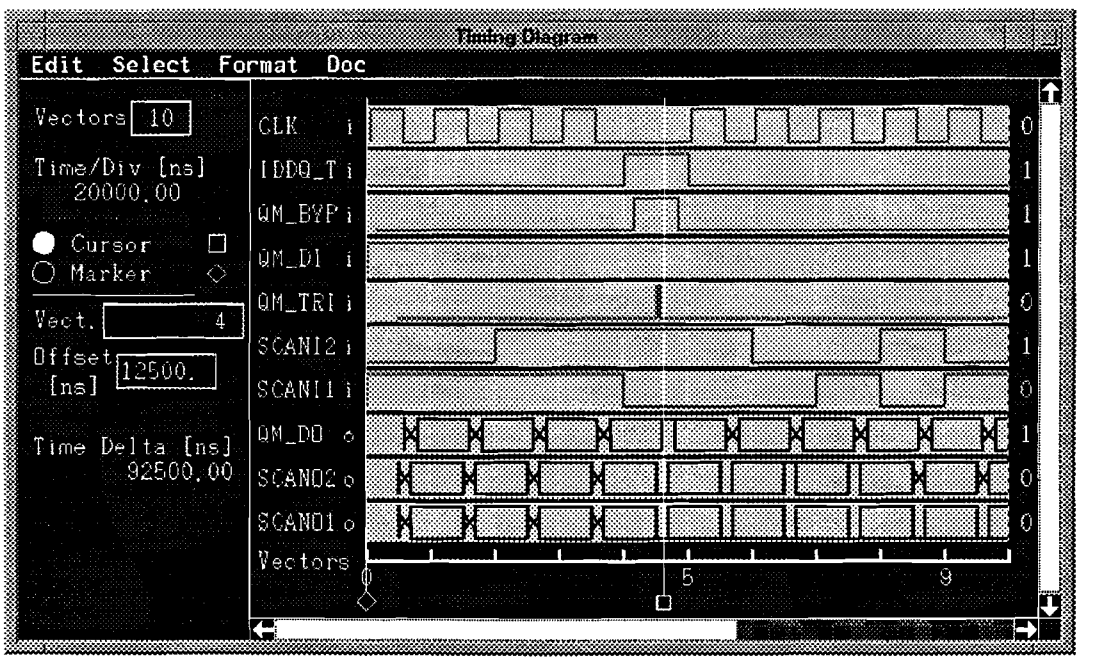


Figure 8-10: HP82000 timing diagram : test DUT with IDUNA-2

9 Conclusions

In this report the definition of a description language for I_{DDQ} monitors has been described. The language 'Monitor Description Format' (MDF) is based on the VHDL syntax and borrows many concepts from the Boundary Scan Description Language (BSDL) used in IEEE Std 1149.1. A wide variety of monitors can be described using MDF because of the flexible storage of monitor control schemes and the extensive collection of standard pin classes, used to interface with Automatic Test Equipment (ATE) software. The MDF definition has focused on creating a working solution for the monitors which already exist (semi-digital). Since no QTAG standard digital interface has been defined yet, the current MDF definition does not include support for these types of monitors. However, this is an area that needs to be addressed in future development.

While defining the I_{DDQ} test-development flow using the Philips test assembler TASS, a timing issue appeared that probably will be a problem with most test assembler software. In conventional voltage test-development, timing information is added at the end of the test generation after assembling the DUT test patterns. In future I_{DDQ} test-development timing information will be more and more important during test vector assembly. One key issue is keeping the DUT quiescent while controlling I_{DDQ} measure equipment. This can be implemented as period changes 'on the fly'. For obtaining optimal testtime vector periods will only be extended in the period I_{DDQ} is measured leaving all other test cycles unchanged.

Another issue identified is the need of a I_{DDQ} trigger column in the DUT test vectors. In this column a flag can be set when I_{DDQ} has to be measured. In Philips I_{DDQ} Test-development flow a virtual pin (Iddq_Trig) is added while assembling the generated DUT testpatterns. This virtual pin is used for triggering I_{DDQ} measurement. As a result, all patterns in the vector files have to be I_{DDQ} testpatterns. Ideal would be pattern generators which generate an I_{DDQ} trigger column in the testpatterns. In this case I_{DDQ} testpatterns could be combined with voltage testpatterns and the QTAG 'Plug & Play' principle would be served because of the automated availability of I_{DDQ} trigger information.

A total 'Plug & Play' environment for QTAG I_{DDQ} monitors only can be reached when ATE offers QTAG standard functions. These functions form together with the QTAG pin classes the software interface needed to control the monitor via the ATE testprogram. For instance, control of PSU voltages is only possible via the testprogram. The QTAG standard functions has to be defined in the near future to allow developments in testprogram generation.

In general can be concluded that the definition of MDF brings the QTAG standard one step closer to a total 'Plug & Play' environment for I_{DDQ} testing although much work has to be done in future development to ensure its success.

10 Future development

In section 6.8 of this report a proposal for a fully digital serial interface is described. This interface uses only 4 pins and is specially designed for QTAG monitors. As a result, fast mode switching (monitor mode / control mode) is possible and also monitor specific settings (e.g. threshold) can be made. Prior to finishing this report a document appeared in which a I_{DDQ} monitor was presented using the standard 1149.1 interface [Rubio, 95].

The 1149.1 standard brings with it the fixed architecture of the TAP controller, a fixed four or five pin interface and BSDL as a definition language. These are all the significant advantages to 1149.1. However, it also constraints the 1149.1 interface. For example the standard defines the state-machine of the TAP controller. That means that fully compliant monitors could not be provided with options to allow rapid changes of operating mode. e.g. in one or two vectors to switch from control mode to calibrate mode, and then bypass mode without additional pins. The pin interface is also tightly constrained.

At first sight the proposed QTAG serial interface is most likely to become standard QTAG digital interface. If both interface types had to be developed from scratch this would be true. Reality shows that 1149.1 is a wide accepted test standard. This means, hardware is available in standard libraries and also test-synthesis tools are available. The MDF defined in this report can perfectly be combined with BSDL. This combination will describe semi-digital as well as digital monitors with 1149.1 interface. In the future maybe digital QTAG monitors can also be combined with boundary scan used for 'on line' current measurements, either on systems or ICs. These benefits make the 1149.1 interface an extremely good candidate to become standard digital interface in these applications. Prior to adopting this interface investigation has to take place to the need of additional pins (monitor clock pin in [Rubio, 95]) in the digital QTAG interface standard for triggering the I_{DDQ} measurements. Furthermore, consequences of the fixed TAP controller should be considered by QTAG.

Appendix A BNF conventions

The syntax of MDF is presented in Backus-Naur Form (BNF)
using the following conventions:

- Any item enclosed in chevrons (i.e., between the character < and the character >) is the name of a syntax item that will be defined.
- Items enclosed between braces (i.e., between the character { and the character }) may be omitted or included one or more times.
- Items enclosed between square brackets (i.e., between the character [and the character]) may be omitted or included only one time.
- Items enclosed between the "lazy T" characters |- and -| may appear in any order.
- Except with regard to case, text shown in **bold** has to be included exactly as it is presented.
- Alternative syntaxes are separated by a vertical bar (|)
- The symbol "::=" should be read as "is defined as".
- White space (spaces, tabulation, carriage returns, etc.) is used in the BNF descriptions to provide enhanced readability and are not part of the syntax.

Appendix B MDF entity description in BNF

• The MDF entity description

entity <component name> **is**
<generic parameters>
<logical port description>
<standard use statement>
{<use statement>}
<monitor type>
<device package pin mappings>
<device port format mappings>
<device default mode mappings>
<monitor port identification>
end <component name>;

<component name>::= <VHDL identifier>

• Generic parameters statement

<generic parameters>::= **generic** (**PHYSICAL_PIN_MAP:string:**=<default package type>
;
PORT_FORMAT_MAP:string:=<default format description>
;
BYPASS_MODE_MAP:string:=<default bypass mode map>
;
MONITOR_MODE_MAP:string:=<default monitor mode map>);

<default package type>::= "<VHDL identifier>"
<default format description>::= "<VHDL identifier>"
<default bypass mode map>::= "<VHDL identifier>"
<default monitor mode map>::= "<VHDL identifier>"

• Logical port description statement

<logical port description>::= **port** (<pin spec> {;<pin spec>});

<pin spec>::= <identifier list> : <pin type><port dimension>
<identifier list>::= <port name>{,<port name>}
<pin type>::= **in** | **out** | **buffer** | **inout** | **linkage**
<port dimension>::= **bit** | **bit_vector** (<range>)
<range>::= <integer> **to** <integer> | <integer> **downto** <integer>

• Standard Use statement

<standard use statement>::= **use** <standard VHDL package identifier>**.all;**

<standard VHDL package identifier>::= **STD_QTAG_1_1995** | <other package identifier>
<other package identifier>::= <VHDL identifier>

- **Use statement**

<use statement>::= **use** <user VHDL package identifier>**.all;**

<user VHDL package identifier>::= <VHDL identifier>

- **Monitor type statement**

<monitor type>::= <monitor operation stmt>
<monitor control stmt>
<monitor return stmt>
[<monitor threshold stmt><monitor threshold value stmt>|
<monitor threshold stmt><monitor threshold limits range stmt>]
[<monitor measurement limits range stmt>]
<monitor minimum clockperiod>

<monitor operation stmt>::=	**attribute MONITOR_OPERATION of** <component name> **:entity is** <monitor operation type>;
<monitor control stmt>::=	**attribute MONITOR_CONTROL of** <component name> **:entity is** <monitor control type>;
<monitor return stmt>::=	**attribute MONITOR_RETURN of** <component name> **:entity is** <monitor return type>;
<monitor threshold stmt>::=	**attribute MONITOR_THRESHOLD of** <component name> **:entity is** <monitor threshold type>;
<monitor threshold value stmt>::=	**attribute MONITOR_THRESHOLD_VALUE of** <component name>**:entity is** <real number>;
<monitor threshold limits range stmt>::=	**attribute MONITOR_THRESHOLD_LIMITS_RANGE of** <component name>**:entity is** (<limits range record>);
<monitor measurement limits range stmt>::=	**attribute MONITOR_MEASUREMENT_LIMITS_RANGE of** <component name>**:entity is** (<limits range record>);
<monitor minimum clockperiod>::=	**attribute MONITOR_MIN_CLOCKPERIOD of** <component name>**:entity is** <real number>;

<monitor operation type>::=	**IDDQ | ISSQ | IXXQ**
<monitor control type>::=	**SEMI_DIGITAL | FULL_DIGITAL**
<monitor return type>::=	**MEASUREMENT | THRESHOLD | BOTH**
<monitor threshold type>::=	**FIXED | VARIABLE**
<limits range record>::=	<lower limit>,<upper limit>
<lower limit>::=	<real number>
<upper limit>::=	<real number>

- **Package pin mapping**

<device package pin mappings>::= <pin map statement><pin mappings>

<pin map statement>::= **attribute PIN_MAP of** <component name>**: entity is PHYSICAL_PIN_MAP;**

<pin mappings>::=	<pin mapping>{<pin mapping>}
<pin mapping>::=	constant <pin mapping name>**:PIN_MAP_STRING** := <pin map string>;
<pin mapping name>::=	<VHDL identifier>
<pin map string>::=	"<port pin map>{,<port pin map>}"
<port pin map>::=	<port name>:<pin list>
<pin list>::=	<pin ID> | (<pin ID>{,<pin ID>})
<pin ID>::=	<integer>

• Port format mapping

<device port format mappings>::= <format map statement><format mappings>

<format map statement>::= **attribute FORMAT_MAP of** <component name>
:entity is PORT_FORMAT_MAP;
<format mappings>::= <format mapping>{<format mapping>}
<format mapping>::= **constant** <format mapping name>
:FORMAT_MAP_STRING:=<format map string>;
<format mapping name>::= <VHDL identifier>
<format map string>::= "<port format map>{,<port format map>}"
<port format map>::= <port name>:<port format>
<port format>::= (<format type>[,<leading edge>][,<trailing edge>])
<format type>::= <digital format type> | <analog strobe type>
<digital format type>::= **DA | RZ | RO | RC | NR | SB | DC | TR**
<analog strobe type>::= **MV**
<leading edge>::= <integer>
<trailing edge>::= <integer>

• Device default mode mapping

<device default mode mappings>::= <bypass mode mappings> <monitor mode mappings>

<bypass mode mappings>::= <bypass mode map statement><mode mappings>
<measure mode mappings>::= <monitor mode map statement><mode mappings>
<bypass mode map statement>::= **attribute BYPASS_MODE of** <component name>
:entity is BYPASS_MODE_MAP;
<measure mode map statement>::= **attribute MONITOR_MODE of** <component name>
:entity is MONITOR_MODE_MAP;
<mode mappings>::= <mode mapping>{<mode mapping>}
<mode mapping>::= **constant** <mode mapping name> : <mode map type> := <mode map string>;
<mode mapping name>::= <VHDL idenifier>
<mode map type>::= **BYPASS_MODE_MAP_STRING | MONITOR_MODE_MAP_STRING**
<mode map string>::= "<port mode map>{,<port mode map>}"
<port mode map>::= <port ID>:<pin state>
<pin state>::= <digital state> | <analog value>
<digital state>::= **0 | 1 | L | H | X | Z**
<analog value>::= <real number>

• Monitor port identification

<monitor port identification>::= | [<bypass>][<trigger>][<di>][<do>][<vdd_psu>]
[<vdd_mon>][<vdd_ref>][<vdd_dut>][<ixxq_ref>]
[<ixxq_ref_bus>][<val_analog>][<vss>] |

<bypass>::= **attribute QMON_BYPASS** **of** <port name>**:signal is** (<level record>);
<trigger>::= **attribute QMON_TRIGGER** **of** <port name>**:signal is** (<level record>);
<di>::= **attribute QMON_DI** **of** <port name>**:signal is** (<level record>);
<do>::= **attribute QMON_DO** **of** <port name>**:signal is** (<level record>);
<vdd_psu>::= **attribute QMON_VDD_PSU** **of** <port name>**:signal is** "<vdd rec str>";
<vdd_mon>::= **attribute QMON_VDD_MON** **of** <port name>**:signal is** "<vdd rec str>";
<vdd_ref>::= **attribute QMON_VDD_REF** **of** <port name>**:signal is** "<vdd rec str>";
<vdd_dut>::= **attribute QMON_VDD_DUT** **of** <port name>**:signal is** "(<dep rec str>)";

<ixxq_ref>::= **attribute QMON_IXXQ_REF** **of** <port name>:**signal is** (<ref record>);
<ixxq_ref_bus>::= **attribute QMON_IXXQ_REF_BUS** **of** <port name>:**signal is** (<ref bus record>);
<val_analog>::= **attribute QMON_VALUE_ANALOG** **of** <port name>:**signal is** "<val_ana rec str>";
<vss>::= **attribute QMON_VSS** **of** <port name>:**signal is** <default value>;

<level record>::= <low level voltage>,<high level voltage>
<low level voltage> ::= <real number>
<high level voltage>::= <real number>

<vdd rec str>::= <min value depend>,<max value depend>,<default value>
<min value depend>::= <real number> | (<dep rec str>)
<max value depend>::= <real number> | (<dep rec str>)
<default value>::= <real number>
<dep rec str>::= <port name>,<relation>
<relation>::= <offset>,<multiplier>
<offset>::= <real number>
<multiplier>::= <real number>

<dep rec str>::= <port name>,<relation>
<relation>::= <offset>,<multiplier>
<offset>::= <real number>
<multiplier>::= <real number>

<ref record>::= <min value>,<max value>,<default value>,<conversion>
<min value>::= <real number>
<max value>::= <real number>
<default value>::= <real number>
<conversion>::= <offset>,<multiplier>
<offset>::= <real number>
<multiplier>::= <real number>

<ref bus record>::= <level record>,<msb ID>,<lsb ID>,<ref table string>
<level record>::= <low level voltage>,<high level voltage>
<low level voltage>::= <real number>
<high level voltage>::= <real number>
<msb ID>::= <integer>
<lsb ID>::= <integer>
<ref table string>::= "<table entry>{,<table entry>}"
<table entry>::= <bus pattern>:<ref value>
<bus pattern>::= <bit>{<bit>}
<bit>::= **0** | **1**
<ref value>::= <real number>

<val_ana rec str>::= <min value depend>,<max value depend>,<conversion>
<min value depend>::= <real number> | <depend record>
<max value depend>::= <real number> | <depend record>
<depend record>::= <port name>,<relation>
<relation>::= <offset>,<multiplier>
<offset>::= <real number>
<multiplier>::= <real number>
<conversion>::= <offset>,<multiplier>
<offset>::= <real number>
<multiplier>::= <real number>

<default value>::= <real number>

Appendix C MDF standard package

```
package STD_QTAG_1_1995 is

-- Give pin mapping declarations

attribute PIN_MAP : string;
subtype PIN_MAP_STRING is string;

-- Give format mapping declarations

attribute FORMAT_MAP : string;
subtype FORMAT_MAP_STRING is string;

-- Give mode mapping declarations

attribute BYPASS_MODE_MAP  : string;
subtype   BYPAS_MODE_MAP_STRING is string;

attribute MONITOR_MODE_MAP : string;
subtype   MONITOR_MODE_MAP_STRING is string;

-- Give monitor_type declarations

type LIMITS_RANGE is record
        LOWER_LIMIT      : real;
        UPPER_LIMIT      : real;
end record;

type MONITOR_OPERATION_TYPE     is (IDDQ,ISSQ,IXXQ);
type MONITOR_CONTROL_TYPE       is (SEMI_DIGITAL,FULL_DIGITALL);
type MONITOR_RETURN_TYPE        is (MEASUREMENT,THRESHOLD,BOTH);
type MONITOR_THRESHOLD_TYPE     is (FIXED,VARIABLE);

attribute MONITOR_OPERATION                    : MONITOR_OPERATION_TYPE;
attribute MONITOR_CONTROL                      : MONITOR_CONTROL_TYPE;
attribute MONITOR_RETURN                       : MONITOR_RETURN_TYPE;
attribute MONITOR_THRESHOLD                    : MONITOR_THRESHOLD_TYPE;
attribute MONITOR_THRESHOLD_VALUE              : real;
attribute MONITOR_THRESHOLD_LIMITS_RANGE       : LIMITS_RANGE;
attribute MONITOR_MEASUREMENT_LIMITS_RANGE: LIMITS_RANGE;
attribute MONITOR_MIN_CLOCKPERIOD              : real;

-- Give monitor identification declarations

type LEVEL_INFO is record
        LEVEL_LOW  : real;
        LEVEL_HIGH : real;
end record;

type REFERENCE_INFO is record
        MIN              : real;
        MAX              : real;
        TYPICAL          : real;
```

```
        OFFSET              : real;
        MULTIPLIER          : real;
end record;

type BUS_REFERENCE_INFO is record
        LEVEL_LOW           : real;
        LEVEL_HIGH          : real;
        MSB_ID              : integer;
        LSB_ID              : integer;
        REF_TABLE_STRING    : string;
end record;

attribute QMON_BYPASS           : LEVEL_INFO;
attribute QMON_TRIGGER          : LEVEL_INFO;
attribute QMON_DI               : LEVEL_INFO;
attribute QMON_DO               : LEVEL_INFO;
attribute QMON_VSS              : real;
attribute QMON_VDD_DUT          : boolean;
attribute QMON_VDD_PSU          : real;
attribute QMON_VDD_MON          : real;
attribute QMON_VDD_REF          : REFERENCE_INFO;
attribute QMON_IXXQ_REF         : REFERENCE_INFO;
attribute QMON_IXXQ_REF_BUS     : BUS_REFERENCE_INFO;

end STD_QTAG_1_1995;
```

Appendix D Lexical elements of MDF (and BSDL)

- **Characterset**

The language is not case sensitive.
The following characters are permitted within the language:

  - Upper- and lower-case letters
  - Digits: **0** to **9**
  - Special characters: **" & ' ( ) * , - . : ; < = > _** (This list is smaller than for VHDL)
  - Logical separators: The space character and VHDL format effectors are used as logical separators. VHDL format effectors are the characters called horizontal tabulation, vertical tabulation, carriage return, line feed and form feed.

- **Identifiers**

Identifiers are user-supplied names and reserved words functioning as names. Identifiers have to start with a letter and may contain letters, digits, or the underscore character.

There is no upper limit to the number of characters in an identifier.
The underscore character (_) is not allowed to be the last character in an identifier.
Adjacent underscore characters (_ _)are not allowed.

Example: *valid_identifier01*
*invalid_identifier_*

- **Strings**

A string is defined as a sequence of zero or more characters enclosed between quotation marks. A quotation mark character is not allowed within a string in MDF (and BSDL).

Example: *"This is a valid string"*
"This "is" not allowed"

Strings are used extensively in MDF (and BSDL). Since many of the strings are potentially longer than a single line, the concatenation operator **&** is used to break them into manageable pieces.

Example: *"This is " & "a split string"*

- **Lexical Atoms**

  - An <integer> is any valid unsigned VHDL integer made up of an unsigned numeric character sequence not containing an underscore character and not using an exponent field
  - A <real number> is any valid VHDL real number of the form <integer>.<integer> or <integer>.<integer>E<integer> all written contiguously without spaces or format effectors.

Note 1E3 is not a real because it does not contain a decimal point.

Appendix E Files MDF2TASS example

E.1 PAT file: 'testdut.pat'

```
Identification;
     Date 20/01/95;
     Time 09:12:13;
     Origin AMSAL;
     Version 01.08.00p;
     Testtype SCANTEST;
     Device DUT;
End;

Header Definition;
     INPUT scani1 1, scani2 2;
     OUTPUT scano1 15, scano2 16;
     CHAIN scani1 3, scano1 9, 4, N;
     CHAIN scani2 7, scano2 13, 2, N;
End;

Pattern Definition;
     1 : 01 1111 11 LLLL HH LL;
     2 : 10 0101 10 HHLH LH HH;
End;
```

E.2 CTR file: 'testdut.tass'

```
DEFINE MULTIPLE;

SET INPUTPIN = scani1, scani2, clk, Iddq_Trig;
SET OUTPUTPIN = scano1, scano2;

Iddq_Trig           = DRIVER TG_NR_TRIG;
scani1,scani2       = DRIVER TG_NR;
clk                 = DRIVER TG_RZ;
scano1,scano2       = STROBE TG_SB;

clk = 1 ;

DEFINE FORMAT TEST_FORMAT;
     Iddq_Trig      = NR;
     scani1,scani2  = NR;
     clk            = RZ;
     scano1, scano2 = SB;
END;

DEFINE TIMING TEST_TIMING;
     SET PERIOD 100 NSEC;
     CHANGE TG_NR_TRIG AT 0;
     CHANGE TG_NR AT 0;
     CHANGE TG_RZ  AT 5, 55;
     CHANGE TG_SB  AT 50, 70;
END;

DEFINE TEST testdut/pat=testdut.pat ;
     SELECT TIMING TEST_TIMING;
     SELECT FORMAT TEST_FORMAT;

     define testmode;
          Iddq_Trig = 0;
     end;
     define normalmode;
          Iddq_Trig = 1;
     end;
END;

CREATE atf no_compress;
END;
```

E.3 ATF file: 'testdut.atf'

```
* ************************************************************************
* * AMSAL PAT file                                                       *
* *                                                                      *
* * Created by  TASS version 01.08.03p                                  *
* ************************************************************************

IDENTIFICATION;
     ORIGIN   TASS        ;
     VERSION  01.08.03p   ;
     DATE     11/04/95;
     TIME     45/36/13;
     TESTTYPE FUNCTIONALTEST;
END;

HEADER DEFINITION;
     INPUT     SCANI1 1, SCANI2 2, CLK 3, IDDQ_TRIG 4;
     OUTPUT    SCANO1 5, SCANO2 6;
END;

PATTERN DEFINITION;

*  ***** START OF 1.TESTMODE   FOR DTB TESTDUT *****
     1 : 1010UU;
     2 : 1010UU;
     3 : 1110UU;
     4 : 1110UU;
*  ***** START OF 1.NORMALMODE FOR DTB TESTDUT *****
     5 : 0111LL;
*  ***** START OF 2.TESTMODE   FOR DTB TESTDUT *****
     6 : 0010LH;
     7 : 1010LH;
     8 : 0110LU;
     9 : 1010LU;
*  ***** START OF 2.NORMALMODE FOR DTB TESTDUT *****
     10: 1011HH;
*  ***** START OF 3.TESTMODE   FOR DTB TESTDUT *****
     11: 0010HL;
     12: 0010HH;
     13: 0010LU;
     14: 0010HU;
END;
```

E.4 CTR file: 'mdf2tass.tass'

```
DEFINE MULTIPLE;

SET INPUTPIN   = QM_DI;
SET INPUTPIN   = QM_TRIGGER;
SET INPUTPIN   = QM_BYPASS;
SET OUTPUTPIN  = QM_DO;

QM_DI          = DRIVER TG_QM_D6;
QM_TRIGGER     = DRIVER TG_QM_D7;
QM_BYPASS      = DRIVER TG_QM_D8;
QM_DO          = STROBE TG_QM_S9;

SET INPUTPIN   = SCANI1,SCANI2,CLK,IDDQ_TRIG;
SET OUTPUTPIN  = SCANO1,SCANO2;

IDDQ_TRIG      = DRIVER TG_NR_TRIG;
SCANI1 ,SCANI2 = DRIVER TG_NR;
CLK            = DRIVER TG_RZ;
SCANO1 ,SCANO2 = STROBE TG_SB;

DEFINE FORMAT MDF2TASS_FORMAT;
     SCANI1    = NR;
     SCANI2    = NR;
     CLK       = RZ;
     IDDQ_TRIG = NR;
     SCANO1    = SB;
```

```
     SCANO2     = SB;
     QM_DI      = NR;
     QM_TRIGGER= NR;
     QM_BYPASS = RZ;
     QM_DO      = SB;
END;

DEFINE TIMING MDF2TASS_TIMING;
     SET PERIOD 20000 NSEC;

     CHANGE TG_NR_TRIG    AT 0;
     CHANGE TG_NR         AT 0;
     CHANGE TG_RZ         AT 1000, 11000;
     CHANGE TG_SB         AT 10000, 14000;
     CHANGE TG_QM_D6      AT 0;
     CHANGE TG_QM_D7      AT 0;
     CHANGE TG_QM_D8      AT 3000, 17000;
     CHANGE TG_QM_S9      AT 10000, 16000;
END;

CREATE HP82000;

DEFINE TEST MDF2TASS_IDDQ/TDL=MDF2TASS.TDL;
     SELECT TIMING MDF2TASS_TIMING;
     SELECT FORMAT MDF2TASS_FORMAT;
END;

END;
```

E.5 TDL file: 'mdf2tass.tdl'

```
VAR pnbr   : integer;
term_nr    : integer;
cycle      : integer;
pins       : integer;

TERM
     { DUT terminals incl. virtual Iddq_Trig pin }
     SCANI1     : INPUT;
     SCANI2     : INPUT;
     CLK        : INPUT;
     IDDQ_TRIG  : INPUT;
     SCANO1     : OUTPUT;
     SCANO2     : OUTPUT;

     { QTAG monitor terminals }
     QM_DI      : INPUT;
     QM_TRIGGER: INPUT;
     QM_BYPASS  : INPUT;
     QM_DO      : OUTPUT;

FUNCTION monitor_bypass_mode;
BEGIN
     DRIVE QM_DI          = &1;
     DRIVE QM_TRIGGER     = &1;
     DRIVE QM_BYPASS      = &1;
     EXPECT QM_DO         = &X;
END;

FUNCTION monitor_measure_mode;
BEGIN
     DRIVE QM_DI          = &1;
     DRIVE QM_TRIGGER     = &1;
     DRIVE QM_BYPASS      = &1;
     EXPECT QM_DO         = &1;
END;

FUNCTION get_quiescent_dut;
BEGIN
     DRIVE CLK = &0;
```

```
     EXPECT SCAN01 = &U;
     EXPECT SCAN02 = &U;
END;

FUNCTION get_pattern_dut;
BEGIN
     for term_nr = 1 to pins do
      set_term term_nr = get_pattern(pnbr,get_patloc(term_name(term_nr)),1);
     endfor;
END;

BEGIN
     pins = 10;
     pnbr = 1;
     open(pat,'testdut.atf');

     writeln(pat_file,'***** TESTMODE ******** *****');
     while read_pattern(pnbr) do

     get_pattern_dut;

     if get_pattern(pnbr,get_patloc('Iddq_Trig'),1) = &0 then
     { standard sycles }
          monitor_bypass_mode;
          for cycle = 1 to get_cycles(pnbr) do
           pattern;
          endfor;
     else
     { measure cycles }
          writeln(pat_file,'***** IDDQ MEASURE MODE *****');
          monitor_measure_mode;
          get_quiescent_dut;
          pattern;
          get_pattern_dut;
          monitor_bypass_mode;
          DRIVE Iddq_Trig = &0;
          writeln(pat_file,'***** NORMALMODE ****** *****');
          pattern;
          writeln(pat_file,'***** TESTMODE ******** *****');
     endif;

     pnbr = pnbr + 1;

     endwhile;
END;
```

E.6 ATF file: 'mdf2tass.atf'

```
* ************************************************************************
* * AMSAL PAT file                                                       *
* *                                                                      *
* * Created by  TASS version 01.08.03p                                  *
* ************************************************************************

IDENTIFICATION;
     ORIGIN   TASS        ;
     VERSION  01.08.03p   ;
     DATE     12/04/95;
     TIME     36/18/15;
     TESTTYPE FUNCTIONALTEST;
END;

HEADER DEFINITION;
     INPUT     QM_DI 1, QM_TRIGGER 2, QM_BYPASS 3, SCANI1 5, SCANI2 6,
               CLK 7, IDDQ_TRIG 8;
     OUTPUT    QM_DO 4, SCANO1 9, SCANO2 10;
END;

PATTERN DEFINITION;
* ***** TESTMODE ******** *****
     1 : 100U1010UU;
     2 : 100U1010UU;
```



```
      3 : 100U1110UU;
      4 : 100U1110UU;
* ***** IDDQ MEASURE MODE *****
      5 : 111H0101UU;
* ***** NORMALMODE ****** *****
      6 : 111U0110LL;
* ***** TESTMODE ******** *****
      7 : 100U0010LH;
      8 : 100U1010LH;
      9 : 100U0110LU;
      10: 100U1010LU;
* ***** IDDQ MEASURE MODE *****
      11: 111H1001UU;
* ***** NORMALMODE ****** *****
      12: 100U1010HH;
* ***** TESTMODE ******** *****
      13: 100U0010HL;
      14: 100U0010HH;
      15: 100U0010LU;
      16: 100U0010HU;
END;
```

Appendix F Examples MDF

F.1 MDF: IDUNA-2

```
entity IDUNA2 is

generic ( PHYSICAL_PIN_MAP: string := "IDUNA2_SOL16";
          PORT_FORMAT_MAP : string := "SYNCHR_FORMAT";
          BYPASS_MODE_MAP : string := "SYNCHR_STATIC_BYPASS";
          MONITOR_MODE_MAP: string := "SYNCHR_ASYNCH_MONITOR");

port (    VREF,VSS,VDD_MON,IREF,VDD_DUT,VDD_PSU:linkage bit;
          DI,TRIGGER,BYPASS:in bit; DO:out bit);

use STD_QTAG_1_1995.all;

attribute MONITOR_OPERATION        of IDUNA2 : entity is IDDQ;
attribute MONITOR_CONTROL          of IDUNA2 : entity is SEMI_DIGITAL;
attribute MONITOR_RETURN           of IDUNA2 : entity is THRESHOLD;
attribute MONITOR_THRESHOLD        of IDUNA2 : entity is FIXED;
attribute MONITOR_THRESHOLD_VALUE  of IDUNA2 : entity is 10.0E-6;
attribute MONITOR_MIN_CLOCKPERIOD  of IDUNA2 : entity is 20.0E-6;

attribute PIN_MAP of IDUNA2:entity is PHYSICAL_PIN_MAP;

constant IDUNA2_SOL16:
          PIN_MAP_STRING:="VREF:2,DO:1,DI:15,VSS:14,VDD_MON:13," &
          "TRIGGER:11,IREF:9,BYPASS:7,VDD_DUT:5,VDD_PSU:3";

attribute FORMAT_MAP of IDUNA2:entity is PORT_FORMAT_MAP;

constant ASYNCH_FORMAT:
          FORMAT_MAP_STRING:="DO:(SB,50,80),DI:(NR,0),TRIGGER:(NR,0)," &
          "BYPASS:(RZ,15,85)";
constant SYNCHR_FORMAT:
          FORMAT_MAP_STRING:="DO:(SB,60,80),DI:(NR,0),TRIGGER:(RZ,50,55)," &
          "BYPASS:(RZ,15,85)";

attribute BYPASS_MODE of IDUNA2:entity is BYPASS_MODE_MAP;

constant ASYNCH_DYNAMIC_BYPASS:
          BYPASS_MODE_MAP_STRING:="DI:1,TRIGGER:1,BYPASS:1,DO:X";
constant SYNCHR_DYNAMIC_BYPASS:
          BYPASS_MODE_MAP_STRING:="DI:1,TRIGGER:0,BYPASS:1,DO:X";
constant ASYNCH_STATIC_BYPASS:
          BYPASS_MODE_MAP_STRING:="VDD_PSU:5.0,DI:1,TRIGGER:1,BYPASS:0,DO:X";
constant SYNCHR_STATIC_BYPASS:
          BYPASS_MODE_MAP_STRING:="VDD_PSU:5.0,DI:1,TRIGGER:0,BYPASS:0,DO:X";

attribute MONITOR_MODE of IDUNA2:entity is MONITOR_MODE_MAP;

constant SYNCHR_ASYNCH_MONITOR:
          MONITOR_MODE_MAP_STRING:="DI:1,TRIGGER:1,BYPASS:1,DO:H";

attribute QMON_BYPASS    of BYPASS : signal is (0.0,5.0);
attribute QMON_TRIGGER   of TRIGGER: signal is (0.0,5.0);
attribute QMON_DI        of DI     : signal is (0.0,5.0);
attribute QMON_DO        of DO     : signal is (0.0,5.0);
attribute QMON_IXXQ_REF  of IREF   : signal is
                                     (-18.0E-6,140.0E-6,0.0,10.0E-6,1.0);
attribute QMON_VDD_PSU   of VDD_PSU: signal is "(VREF,0.0,1.0),6.5,5.5";
attribute QMON_VDD_MON   of VDD_MON: signal is "(VREF,0.0,1.0),6.5,5.5";
attribute QMON_VDD_REF   of VREF   : signal is "(VDD_PSU,-0.5,1.0),6.5,5.0";
attribute QMON_VDD_DUT   of VDD_DUT: signal is "(VREF,0.0,1.0)";
attribute QMON_VSS       of VSS    : signal is 0.0;

end IDUNA2;
```

F.2 MDF: QUICKMON V5.0a

```
entity QUICMON_V5_0_A_ISSQ is

generic ( PHYSICAL_PIN_MAP : string := "QUICMON_V5_0_A_ISSQ_BOARD";
          PORT_FORMAT_MAP  : string := "QUICMON_V5_0_A_ISSQ_FORMAT";
          BYPASS_MODE_MAP  : string := "QUICMON_V5_0_A_ISSQ_BYPASS";
          MONITOR_MODE_MAP : string := "QUICMON_V5_0_A_ISSQ_MONITOR");


port (    POS15V,VSS,POS5V,NEG15V,SENSE:linkage bit;
          CHG,BYPSS:in bit);

use STD_QTAG_1_1995.all;

attribute MONITOR_OPERATION                  of QUICMON_V5_0_A_ISSQ:
                                             entity is ISSQ;
attribute MONITOR_CONTROL                    of QUICMON_V5_0_A_ISSQ:
                                             entity is SEMI_DIGITAL;
attribute MONITOR_RETURN                     of QUICMON_V5_0_A_ISSQ:
                                             entity is MEASUREMENT;
attribute MONITOR_MEASUREMENT_LIMITS_RANGE   of QUICMON_V5_0_A_ISSQ:
                                             entity is (0.0,0.0);
attribute MONITOR_MIN_CLOCKPERIOD            of QUICMON_V5_0_A_ISSQ:
                                             entity is 10.0E-6;

attribute PIN_MAP of QUICMON_V5_0_A_ISSQ:entity is PHYSICAL_PIN_MAP;

constant QUICMON_V5_0_A_ISSQ_BOARD:

PIN_MAP_STRING:="POS15V:1,VSS:2,POS5V:3,NEG15V:4,SENSE:5,CHG:6,BYPSS:7";

attribute FORMAT_MAP of QUICMON_V5_0_A_ISSQ:entity is PORT_FORMAT_MAP;

constant QUICMON_V5_0_A_ISSQ_FORMAT:
          FORMAT_MAP_STRING:="CHG:(RO,50,90),BYPSS:(NR,0),SENSE:(MV,60,85)";

attribute BYPASS_MODE of QUICMON_V5_0_A_ISSQ:entity is BYPASS_MODE_MAP;

constant QUICMON_V5_0_A_ISSQ_BYPASS:
          BYPASS_MODE_MAP_STRING:="BYPSS:0,CHG:1";

attribute MONITOR_MODE of QUICMON_V5_0_A_ISSQ:entity is MONITOR_MODE_MAP;

constant QUICMON_V5_0_A_ISSQ_MONITOR:
          MONITOR_MODE_MAP_STRING:="BYPSS:1,CHG:0";

attribute QMON_BYPASS       of BYPSS  : signal is (0.0,5.0);
attribute QMON_TRIGGER      of CHG    : signal is (-2.0,5.0);
attribute QMON_VSS          of VSS    : signal is 0.0;
attribute QMON_VDD_MON      of POS15V : signal is "15.0,15.0,15.0";
attribute QMON_VDD_MON      of POS5V  : signal is "5.0,5.0,5.0";
attribute QMON_VDD_MON      of NEG15V : signal is "-15.0,-15.0,-15.0";
attribute QMON_VALUE_ANALOG of SENSE  : signal is "0.0,0.0,0.0,0.0";

end QUICMON_V5_0_A_ISSQ;
```


F.3 MDF: LTX_I_{DDQ}

```
entity LTX_IDDQ is

generic ( PHYSICAL_PIN_MAP : string := "LTX_IDDQ_PINS";
          PORT_FORMAT_MAP  : string := "LTX_IDDQ_FORMAT";
          BYPASS_MODE_MAP  : string := "LTX_IDDQ_BYPASS_LIMIT0";
          MONITOR_MODE_MAP : string := "LTX_IDDQ_MONITOR_LIMIT0");

port (    DPS_IN,DPS_OUT:linkage bit;
          GND:linkage bit_vector(0to1);
          CONTROL:inout bit;
          ADR:in bit_vector(0to3));

use STD_QTAG_1_1995.all;

attribute MONITOR_OPERATION              of LTX_IDDQ : entity is IDDQ;
attribute MONITOR_CONTROL                of LTX_IDDQ : entity is SEMI_DIGITAL;
attribute MONITOR_RETURN                 of LTX_IDDQ : entity is THRESHOLD;
attribute MONITOR_THRESHOLD              of LTX_IDDQ : entity is VARIABLE;
attribute MONITOR_THRESHOLD_LIMITS_RANGE of LTX_IDDQ : entity is (0.0,0.0);
attribute MONITOR_MIN_CLOCKPERIOD        of LTX_IDDQ : entity is 10.0E-6;

attribute PIN_MAP of LTX_IDDQ:entity is PHYSICAL_PIN_MAP;

constant LTX_IDDQ_PINS:
          PIN_MAP_STRING:="DPS_IN:1,DPS_OUT:2,GND:(3,4),CON-
TROL:5,ADR:(6,7,8,9)";

attribute FORMAT_MAP of LTX_IDDQ :entity is PORT_FORMAT_MAP;

constant LTX_IDDQ_FORMAT:
          FORMAT_MAP_STRING:="CONTROL:(SB,65,95),CON-
TROL:(RZ,5,60),ADR:(NR,0)";

attribute BYPASS_MODE of LTX_IDDQ :entity is BYPASS_MODE_MAP;

constant LTX_IDDQ_BYPASS_LIMIT0:
          BYPASS_MODE_MAP_STRING:="CONTROL:0,CONTROL:X," &
          "ADR(3):0,ADR(2):0,ADR(1):0,ADR(0):0";

constant LTX_IDDQ_BYPASS_LIMIT4:
          BYPASS_MODE_MAP_STRING:="CONTROL:0,CONTROL:X," &
          "ADR(3):0,ADR(2):1,ADR(1):0,ADR(0):0";

attribute MONITOR_MODE of LTX_IDDQ :entity is MONITOR_MODE_MAP;

constant LTX_IDDQ_MONITOR_LIMIT0:
          MONITOR_MODE_MAP_STRING:="CONTROL:1,CONTROL:H," &
          "ADR(3):0,ADR(2):0,ADR(1):0,ADR(0):0";

constant LTX_IDDQ_MONITOR_LIMIT4:
          MONITOR_MODE_MAP_STRING:="CONTROL:1,CONTROL:H," &
          "ADR(3):0,ADR(2):1,ADR(1):0,ADR(0):0";

attribute QMON_BYPASS       of CONTROL : signal is (0.0,5.0);
attribute QMON_DO           of CONTROL : signal is (0.0,5.0);
attribute QMON_VSS          of GND     : signal is 0.0;
attribute QMON_VDD_DUT      of DPS_OUT : signal is "(VDD_PSU,0.0,1.0)";
attribute QMON_VDD_PSU      of DPS_IN  : signal is "0.0,6.0,5.0";
attribute QMON_IXXQ_REF_BUS of ADR     : signal is (0.0,5.0,3,0,"0000:0.0," &
                                                    "0001:0.0," &
                                                    "0010:0.0," &
                                                    "0011:0.0," &
                                                    "0100:0.0," &
                                                    "0101:0.0," &
                                                    "0111:0.0," &
                                                    "1000:0.0," &
                                                    "1001:0.0," &
                                                    "1010:0.0," &
                                                    "1011:0.0," &
                                                    "1100:0.0," &
                                                    "1101:0.0," &
                                                    "1110:0.0," &
                                                    "1111:0.0");

end LTX_IDDQ;
```

References

[Baker, 90] K. Baker, S.C. Verhelst,
"I_{DDQ} Testing because Zero-defects isn't enough:
A Philips Perspective",
Proceedings International Test Conference 1990, pp 253-254

[Baker, 93] K.Baker,
"Philips' Draft Proposal for a Quiescent Current Monitor
Standard under QTAG"
version 1.0 QTAG, 1993

[Baker, 94a] K. Baker,
"Requirements Specification for the QTAG
Monitor Description Format (MDF)"
QTAG document, version 1.0, May 1994
Internal reference RWB-554-KB-94052-KB

[Baker, 94b] K. Baker, et-al,
"Development of a Class 1 QTAG Monitor",
Proceedings International Test Conference, 1994, pp 213-222

[Baker, 95] K. Baker, A. Hales,
"Quality Test Action Group (QTAG):
Plug and Play I_{DDQ} Testing for Text Fixtures",
Design & Test of Computers (IEEE), FALL, 1995

[Bennetts, 84] R.G. Bennetts,
"Design of testable logic circuits",
Addison-Wesley, 1984

[Bouwman, 92] F.G.M. Bouwman, et-al
"Macro Testability: The Results of Production Device Applications"
Proceedings International Test Conference 1992, pp 232-241

[Bouwman, 95] F.G.M. Bouwman, et-al,
"Performing Application Mode Testing in a DSP based Testing
Environment",
Philips internal report 1995

[Bruls, 94] E. Bruls,
"Variable supply voltage testing for analogue CMOS and
bipolar circuits",
Proceedings International Test Conference 1994, pp. 562-571

[CAT team, 94] CAT team,
"TASS Reference Manual version 01.08.03",
Philips Semiconductors, sept 1994

[Claassen, 89] T.A.C.M. Claassen, et-al,
"New Directions in Electronic Test Philosophy,Strategy, and Tools",
Proceedings of 1st European Test Conf. 1994, Paris 1989, pp. 5-13

[Hales, 94a] A. Hales
"Texas Instrument Draft Proposal for QTAG off-chip I_{DDQ} / I_{SSQ} Monitor Standard",
version 1.4, QTAG, 1994

[Hales, 94b] A. Hales
"A serially Adressable, Flexible Current monitor for Test Fixture based I_{DDQ}/I_{SSQ} Testing",
Proceedings International Test Conference, 1994, pp 223-232

[Keating,87] M. Keating and D. Meyer,
"A New Approach to Dynamic IDD Testing",
Proceedings International Test Conference, 1987, pp 316-321

[LTX, 94] LTX corporation
"Master Series I_{DDQ} monitor"
Preliminary document Rev 0.8 , 1994

[Maly, 88a] W. Maly, P. Nigh,
"Built-in Current Testing Feasability Studie",
IEEE ICCAD, Nov 1988, pp 340-343

[Maly, 88b] W. Maly, et-al,
"Yield loss mechanisms and defect tolerance"
in "Yield modelling and defect tolerance in VLSI" pp 3-30,
Adam Hilger, Philadelphia, PA, USE, 1988

[Manhoeve, 94] H.A.R. Manhoeve, et-al,
"An Off-chip I_{DDQ} Current Measurement Unit for Telecommunications ICs"
Proceedings International Test Conference, 1994

[Maxwall, 92] P.C. Maxwall, et-al,
"The effectiveness of I_{DDQ}, functional and scan tests:
How many fault coverages do we need?",
Proceedings International Test Conference 1992, pp. 168-177

[Parker, 90] K.Parker, S. Oresjo,
"A Language for Describing Boundary-Scan Devices"
Proceedings International Test Conference 1990, pp. 222-231

[Perry, 93] D.L. Perry
"VHDL second edition"
Mc Graw-Hill inc. Computer Engineering Series, 1993
ISBN 0-07-049434-7

[Rajsuman, 95] R.Rajsuman,
"I_{DDQ} testing for CMOS VLSI",
Artech house, 1995

[Rubio, 95] A. Rubio
"I_{DDQ} monitors design strategies"
VLSI Test Symposium, 1995

[Wallquist, 93] K.M. Wallquist, et-al,
"A General Purpose I_{DDQ} Measurement Circuit",
Proceedings International Test Conference 1993, pp 642 - 649

[Wallquist, 94] K.M. Wallquist,
"I_{DDQ}/I_{SSQ} production testing with QuiC-Mon v5.00"
Document, Philips Semiconductors, Albuquerque

