Divide-and-Conquer IDDQ Testing for Core-based System Chips

C.P. Ravikumar
Texas Instruments India Pvt Ltd
Wind Tunnel Road, Murgeshpalya
Bangalore-560017, India
ravikumar@ti.com

Rahul Kumar
Sasken Communication Technologies Ltd.
#21, Brigade Square, Cambridge Road,
Ulsoor, Bangalore-560008
rahulk@sasken.com

Abstract

IDDQ testing has been used as a test technique to supplement voltage testing of CMOS chips. The idea behind IDDQ testing is to declare a chip as faulty if the steady-state current drawn from the power supply after the application of a test vector exceeds a threshold value. A CMOS circuit only consumes leakage power after the switching transients settle down, and a large quiescent power-line current indicates a defective chip. With device counts in system chips crossing into millions, the leakage power is no more insignificant, making IDDQ tests unsafe. Yet, IDDQ tests are invaluable since they can catch faults that are not testable using voltage testing. In this paper, we propose a solution to make IDDQ testing practical for large system chips. Our technique is based on chip partitioning and scheduling the testing of partitions so that IDDQ testing can be safely practiced. We formulate partitioning as a constrained optimization problem and propose two algorithms for partitioning. The objective function for the optimization problem is the test execution time. We present experimental results to illustrate our methodology.

1.0 Introduction

Current testing can be applied to Static CMOS circuits since they draw negligible current under steady state. The quiescent power-line current after the application of a test vector is referred to as IDDQ and consists of subthreshold leakage current and PN-junction leakage current. If there is a defect in the circuit, such as a short circuit between two nodes, a direct resistive path may be formed between VDD and VSS, causing an increase in IDDQ. By suitably selecting test vectors [1] and monitoring IDDQ after the application of the test vectors [2], defective chips can be separated from good ones. Defects such as gate-oxide shorts, which may not change the functional behavior of the circuit, can be detected using IDDQ testing [1,2,5,11]. IDDQ tests are particularly well suited for detecting bridge faults, open transistor faults, transistor stuck-on faults, punch-through faults, PN-junction leakage faults, and abnormally high contact resistances [1,5,11]. Since IDDQ tests detect defects that are not necessarily logic faults, their use improves the circuit reliability and quality.

However, IDDQ testing is becoming difficult to practice for system chips implemented in deep sub-micron technologies. There are many reasons for this problem. The sub-threshold leakage current increases for deep sub-micron technologies. As an illustration, a chip implemented in 0.8 μm technology operating at 5V has a typical oxide thickness of 150 Angstrom, a threshold voltage of 0.8V, and the leakage current is around 0.01 pA/μm. On the other hand, a chip implemented in 0.18 μm technology will operate at a scaled-down voltage of 1.8V, has an oxide thickness of about 50 Angstrom, a threshold voltage of around 0.5V, and a leakage current of about 300 pA/μm [15]. In a nutshell, system chips have a larger number of transistors, each of which draws a larger sub-threshold current. Thus a good chip draws a significant IDDQ and to distinguish it from a defective chip would be difficult and require the use of very powerful current sensors. There are technological solutions to this problem. For instance, since the subthreshold leakage current heavily depends on the operating temperature, one can carry out IDDQ testing at low temperatures to make them viable [12]. Such a solution will increase both the initial cost and operating cost of the test equipment. Other solutions to reduce the leakage current are the use of substrate biasing, multiple threshold CMOS, and SOI technology. In this paper, we propose an alternate solution to the problem, which does not rely on technological enhancements to reduce the leakage current. Our solution relies on partitioning the circuit into blocks so that IDDQ testing can be safely practiced on individual blocks.

We organize the paper as follows. The partitioning approach to IDDQ testing is described in the following section. Estimators for fault-free IDDQ and test execution time for cores and core-based systems are given in Section 3. Two algorithms for performing circuit partitioning and test scheduling are discussed in Section 4. Experimental results are described in Section 5. Conclusions are presented in Section 6.

2.0 Partitioning

As mentioned in the previous section, system chips have large IDDQ due to the large number of devices on the chip, and setting an IDDQ threshold to distinguish faulty chips from good ones is difficult. To circumvent this problem, we force a test plan in which only a portion of the circuit is tested and the other parts of the circuit are powered off. Since SOC design is core-based, the granularity of the partition is up to cores. The essential principle of testing by partitioning and how it improves the confidence in the tested products can be described using an analogy. Suppose the students in a class appear for tests in *n* subjects and their scores in the subjects are

denoted S_1, S_2, …, S_n. In one system of evaluation, the scores are added and a student is declared to have passed if (S_1+S_2+ … + S_n) is larger than a threshold. The confidence of such a procedure is obviously low, since a student who may have scored very low in a subject would pass if he has scored high in the other subjects. Selecting the threshold in such a system of examination is difficult when *n* is large and the maximum score in a subject is large. (Any teacher knows it is easier to grade a student on a scale of 1 to 10 than a scale of 1 to 100.) In an alternate system of evaluation, we could "partition" the aggregate into individual scores and set up separate threshold for each subject. A student is said to pass if his/her score S_j is larger than the threshold selected for subject *j*, for $j = 1, 2, \ldots, n$. We have more confidence in a student who has passed in the second system of evaluation.

The test architecture we select for System-On-Chip IDDQ testing is the IEEE P1500 architecture described in [14]. Currently, the P1500 architecture is suitable for voltage testing.

2.1 Enhancement to P1500 Architecture

Since the modifications to the test architecture must not result in excessive area overhead or have major implications on the DFT issues of the core, we use the technique of high-threshold voltage switch. A similar technique has been used by Rajsuman [6]. A high-threshold switch is added to the core wrapper for isolating the core from the power supply. The gate voltage of the switch can be controlled to turn off the switch, cutting off the power supply to the core to which the switch is connected. A P1500-compliant test architecture is shown in Figure 1. An isolation control register (ICR) selects which cores are powered off during testing. The outputs of the register control the gates of the high-threshold switches. The high-threshold switch can be regarded as part of the core wrapper. The wrapper also consists of scan flip-flops, which, depending on whether they are placed on the input side or the output side, are useful for scanning in test data and scanning out test responses (for voltage testing). A bypass register is useful for isolating the core from the TAM, so that test data can be forwarded to another core.

2.2 Quantitative Analysis

Let there be *n* cores in the system. When current testing is applied individually to each core, let the IDDQ for a fault-free core *j* be given by $IDDQ_j$. The total IDDQ for a fault-free chip is given by

$$IDDQ = IDDQ_1 + IDDQ_2 + \ldots IDDQ_n$$

Note that $IDDQ_j$ are random variables, since the current depends on the operating conditions, input pattern, and the variations in the manufacturing process. Usually, $IDDQ_j$ are taken to be Gaussian random variates. Let μ_j and σ_j be the mean and standard deviation of current $IDDQ_j$. Then the mean μ and standard deviation σ of the cumulative current IDDQ are given by

$$\mu = \mu_1 + \mu_2 + ... + \mu_n$$
$$\sigma^2 = \sigma_1^2 + \sigma_2^2 + \ldots + \sigma_n^2$$

For a faulty chip, the IDDQ can be written as

$$IDDQ_f = IDDQ + I_f$$

where I_f corresponds to the extra current that the SOC will sink due to a resistive path from VDD to VSS. The mean and standard deviations of $IDDQ_f$ are given by

$$\mu_{IDDQf} = \mu_{IDDQ} + \mu_{If}$$
$$\sigma^2_{IDDQf} = \sigma^2_{IDDQ} + \sigma^2_{If}$$

It is common to set the IDDQ threshold to (μ_{IDDQ} + $3\sigma_{IDDQ}$). Due to the intrinsic leakage of the system, the distribution of fault-free IDDQ may overlap with the distribution of the faulty IDDQ (see Figure 2(a)) and the confidence in a tested product is not high due to chances of "aliasing." Now suppose that we partition the set of cores into *k* groups and test each group separately. The threshold limit of (μ_{IDDQ} + $3\sigma_{IDDQ}$) will be applicable to each group, and since the associated mean and standard deviations are small, the chances of "aliasing" are smaller for a subset of cores. Therefore, the confidence in the tested product improves (Figure 2(b)).

3.0 Optimization

We have outlined in the previous section a P1500-compliant IDDQ test architecture. We now formulate partitioning as an optimization problem. Let $C = \{C_1, C_2, \ldots, C_n\}$ be the set of cores in the SOC. Let $P = P_1 \oplus P_2 \oplus \ldots \oplus P_k$ be a partition of C, where P_j are subsets of C such that $P_i \cap P_j = \varphi$ if $i \neq j$, and $P_1 \cup P_2 \ldots P_k = C$. Two extreme cases of partitioning occur when k=1 and k=*n*. In the former case, all cores are in the same partition, and the resulting IDDQ may be too large for reliable testing. In the latter case, the cores are tested one at a time, increasing the TAT. We desire an intermediate solution that minimizes the TAT while ensuring the reliability of the IDDQ test procedure.

Scheduling of the testing of cores on an SOC also has implications on dynamic power consumption during testing; testing too many cores at the same time can burn down the chip [13]. Partitioning the set of cores and scheduling the testing of the partitions under power constraints has been studied in [7,9]. In this work, voltage testing was assumed and the estimation of test execution time is different.

It is not hard to appreciate that the partitioning problem is computationally difficult. The number of solutions to the *k*-way partitioning problem is the Stirling number of the second order, $S(n,k)$, which grows sharply with *n* and *k*. For a system with 16 cores, the number of 10-way partitions is 193754990. To make matters worse, we do not know the number of partitions *k* a priori. Due to the above considerations, we propose heuristic solutions to the partition problem. These are described in the following section.

3.1 Problem Formulation

The inputs to the problem include the description of the system with details such as the number of cores, *n*, the descriptions of the cores, and the upper threshold I_t on the mean value of IDDQ that is acceptable from the viewpoint of reliability. For the purpose of this work, the description of a core j includes items that are mainly useful in evaluating the test execution time for a given core partition and test schedule. These are the number of functional inputs FI_j to the core, the number of scan inputs SI_j to the core, the number of IDDQ test patterns τ_i to the core, and the actual test patterns. We assume that the cores make use of one or more internal scan chains for the purposes of testing. Let the lengths of the scan chains in the core j be denoted $s_j(0)$, $s_j(1)$, …, $s_j(SI_j\text{-}1)$. Let ST_{ji} be the settling time taken by core j after the application of the i^{th} pattern. This is the time for transient current to settle down when inputs stabilize. The outputs include the partition P of the core set C and a test schedule for the cores assigned to the partitions. The objective is to minimize the estimated TAT while maintaining the estimated IDDQ during a test session to be below I_t. We now describe the procedures used to estimate the TAT and the maximum IDDQ.

3.2 Estimation of Test Time

Suppose the partition information $P = P_1 \oplus P_2 \oplus \ldots \oplus P_k$ is known. We estimate the total test execution time as the sum of the test times for all the partitions. Since all the cores assigned to a partition are tested concurrently, the test time for a partition is the maximum of the test execution times of all the cores assigned to the partition. The core test time is calculated as the sum of the time taken to scan in the test patterns and the sum of the settling times of IDDQ after the application of each test pattern. The time taken to scan in a test pattern depends on the number of scan chains, the number of functional inputs to the core, the lengths of the scan chains and the width of the TAM. The following procedure is used to estimate the total scan-in time.

```
procedure ScanInTime(SI,FI,T,ChainLength)
//SI and FI are the number of scan and functional inputs
//T is the TAM width
//ChainLength is an array containing the lengths of scan
chains
begin
        declare A: array [0..SI+FI-1] of integer;
        for (i=0;i<SI;i++) A[i] = ChainLength[i];
        for (i=SI;i<SI+FI;i++) A[i] = 1;
        while (A[SI] > 0) do begin
                A[0] = A[0] + A[SI];
                A[SI] = 0;
                sort(Ascending,A[0..SI-1]);
                sort(Descending,A[SI..SI+FI-1]);
        end
        return(A[SI-1]);
end
```

As an example, suppose the procedure is applied to the s713 benchmark circuit which has 19 flip-flops and 35 functional inputs. Assuming 4 scan chains of lengths 5, 5, 5, and 4, and a TAM width of 3, the above procedure will estimate the scan-in time for a single test pattern to be 18 clock cycles.

In our experimentation, we use the data presented in [1] to estimate the settling times ST_{ji}. Chakravarty and Thadikaran [1] have reported the transient current for a chain of inverters through SPICE simulations. They reported the propagation delay for a 1000-gate deep inverter chain as well as the settling time for the inverter chain, and observed that the settling time can be any where from 1.1 times to 1.93 times the propagation time PDi for the pattern P_i. Thus, pessimistically, one could take the settling time to be twice the propagation delay. Under this assumption, the number of clock cycles required for the current to settle down is $\lceil 2.\ PD_i/T_o \rceil$ where T_o is the clock frequency of the tester.

3.3 Estimation of IDDQ

Estimation of IDDQ is a vast topic and has been studied extensively in the literature. The actual measurement of the IDDQ for a fault-free device is done using an off-chip current sensor [3]. The measurements are carried out for a large sample of chips since IDDQ varies from one instance of the chip to another due to variations in process parameters. Operating conditions, namely voltage and temperature, also affect IDDQ, and measurements have to be repeated for many operating conditions as well. If the average of the measured values is μ and the standard deviation is σ, then the threshold IDDQ is taken to be $\mu + 3\sigma$. Analytical and semi-analytical approaches have also been reported to estimate IDDQ [1,4]. We propose a simple method for leakage power estimation and use it to estimate the threshold IDDQ. We used 35 different benchmark circuits, 8 combinational and 27 sequential benchmarks, and measured the leakage power for each of them using the estimator available in Synopsys DesignCompiler. The TSMC 0.18 μm library was used. Timing constraints and area constraints were fed for each circuit to obtain gate-level realizations of the benchmark circuits. For each circuit, we repeated the estimation for all the three libraries, slow, fast, and typical. A log-log plot of the leakage power Vs the number of cells in the gate-level realization was generated. It was seen that the plot was nearly linear with the parameters of Table 1.

Table 1: Parameters of Log-Log Plot of Leakage Power Vs Number of Cells

Library	Slope of the line	Intercept
Slow	0.966	-1.032
Fast	0.946	-1.633
Typical	0.993	-2.212

Using the above data, we can write down equations for the leakage power as a function of the number of cells; there will be one equation for each library/operating condition. The leakage power for a circuit with a known number of cells and a given library/operating condition can then be estimated through extrapolation. The leakage current is obtained by dividing the leakage power by the operating voltage corresponding to the library. Due to limitation of space, we omit the details of the linear model. We validated the model using the example of the 24-bit CMU DSP core and obtained the results tabulated in Table 2. Percentage error was calculated by synthesizing the DSP circuit using Design Compiler and using the leakage power estimator of the Design Compiler on the gate-level realization. The results shown in Table 2 are for the "slow" library. Since our results indicate that the estimator gives results within 10% error, we have used this technique in our experimentation.

Table 2: Validation of the Leakage Estimation Model

Component	# Gates	Leakage Power (μW)		% Error
AGU	6500	1.59	1.71	8.22
ALU	7548	1.92	1.98	3.12
PCU	3436	0.87	0.92	5.74
Bus Switch	458	0.16	0.14	-12.50
Core	17962	4.53	4.58	1.10

4.0 Algorithms

We propose two algorithms for the partitioning problem, namely, a greedy algorithm and a genetic algorithm. These are now described briefly. In both these algorithms, the partition information was represented using a simple integer array data structure of *n* elements, where *n* is the number of cores. The j^{th} entry in the array indicates the partition to which the core has been assigned. The inputs to our algorithms are two files, the first one containing the description of the system chip, and the second one containing the description of the cores used in the chip. The estimators described in the previous section are used to calculate the IDDQ and test execution time for each core. IDDQ and test execution time for a partition and the system are calculated using the method described in the previous section.

4.1 Greedy Algorithm

The greedy algorithm begins with a randomly generated partition P which meets the reliability constraint; in other words, the estimated IDDQ for the system which will be tested using the partition information P is no greater than the threshold current I_t. Random partitions are repeatedly generated before one such partition is found. The execution time for the partition P is computed. The algorithm then attempts to iteratively improve the partition P by making local transformations to P. In each iteration, a large number of local transformations are attempted, which consist of random perturbations to the partition information. If a transformation improves the test time, the transformation is applied and a new partition is generated. If none of the transformations are able to improve the partition, the algorithm quits the improvement procedure since a "local optimum" has been achieved.

4.2 Genetic Algorithm

Since the greedy algorithm can get stuck in a local optimum solution, we also experimented with a genetic algorithm for the partitioning problem which is known for its potential for yielding global optimum solutions. Genetic algorithms have been applied to a number of problems in VLSI/EDA and Testing [8,10]. Unlike other optimization algorithms, a genetic algorithm starts with a pool of solutions to the problem. Thus, a population of *N* feasible solutions for the partitioning problem are generated randomly. The encoding of each of these solutions is known as a "chromosome" and has a "fitness" index, which is a measure of its optimality. Better solutions have a higher fitness and vice versa. From one population, the next generation of population is obtained through the process of procreation and "survival of fittest." Procreation involves the generation of *N*+*N'* solutions through the use of genetic operators, namely, genetic copy, crossover, and mutation. Genetic copy involves creating a chromosome that is identical to a parent. Crossover is a binary operator which combines the properties of two chromosomes, called the parent chromosomes, into a child chromosome. When selecting parents for crossover, fitter solutions from the parent population are given a higher chance. Mutation involves the introduction of small random changes in a child resulting either from a genetic copy or a crossover operation. "Survival of the Fittest" is the process of eliminating the *N*' least fit solutions from the new generation and sizing the population back to *N*. It is clear that the best solution of the second generation is either better or at least as good as the best solution of the first generation. More generations can be obtained by repeating the above process, until a convergence criterion is satisfied. A possible convergence criterion is that the number of successive generations without improvement in the quality of the best solution crosses a specified limit. Alternately, the evolution process can be terminated after a specified number of generations or when the time for optimization crosses a limit.

In our implementation, we used the integer array encoding explained early in this section to represent a chromosome. The crossover operator is a "cut-and-paste" operation which selects the elements 0..*i* from the chromosome of one parent and elements *i*+1..*n* from the second parent, for a randomly selected i, $1 < i < n$. The mutation operator swaps elements i and j in the child chromosome, where $0 \leq i < j \leq n$, and i,j are selected randomly. Note that although the operators are simple to implement and fast to execute, they may result in solutions that are invalid. For instance, the crossover operator may result in a solution in which some partitions are empty. Consider n=6 and i=3, and the two

parents are 111234 and 233124; in the child chromosome 111124, partition 3 is empty. To correct the problem, we must renumber the partition number 4 as partition number 3. This renumbering requires O(*n*) time. The complexity of the crossover operator is O(*n*). The mutation operator requires O(1) time. Both crossover and mutation operators can yield solutions that are infeasible i.e. estimated IDDQ during testing exceeds I_t. When such is the case, the solutions are assigned low fitness value so that the process of natural selection eliminates them.

5.0 Experimental Results

The algorithms discussed in the previous section were implemented in C programming language on a Pentium PC/II system under Linux operating system. The other tools that were employed for leakage current estimation include SPICE3 from UC Berkeley and Synopsys Design Compiler working on a Sun server under the Unix operating system. Since we did not have access to data for any system chips, we decided to make up our own SOC benchmarks using the well known ISCAS89 sequential benchmark circuits as cores. Our first benchmark SOC consisted of 14 cores and the second SOC had 20 cores. Note that the example circuits do not represent any meaningful chips. . Due to space limitation, we shall only discuss the results for the first SOC. We show the details of the SOC in Table 3. The IDDQ shown in the Column 6 is for the fault-free core. The standard deviation of IDDQ was taken to be 5% of the IDDQ for the respective cores. The faulty IDDQ was calculated by simulating a 4-input NAND gate in SPICE3 with a "short" (50 kΩ) placed across the channel of one of the NMOS transistors.

Table 3: Details of Benchmark SOC

Core	Instances	FI	SI	Vectors	IDDQ (nA)
s5378	1,7	35	10	149	115.33
s9234	2,8,13	36	15	235	91.08
s15850	3,9,14	77	20	292	319.00
s38417	4,10	28	18	262	795.47
s35932	5,11	35	22	52	733.14
s13207	6,12	62	14	233	311.33

The value of I_t was selected to be 5149.91 nA which is the sum of the fault-free IDDQ for all cores i.e. all cores are tested concurrently plus a safety margin of 3σ.

The results obtained by the two algorithms are tabulated in Table 4. We see that the genetic algorithm finds a marginally better solution than the greedy algorithm, although it takes slightly longer to run. Interestingly, both algorithms found a two-partition solution. In the greedy algorithm, in each iteration, 3000 local transformations were attempted. The parameters of the genetic algorithm were selected to be N=2000, N' = 1000, and number of generations = 10. We tuned the genetic algorithm's performance by studying the variation of the fitness function as a function of the number of generations; these results are omitted for brevity. For the second SOC example, the genetic algorithm yielded a partition with a test execution time of 775 clock cycles, as opposed to the 790-cycle solution given by the greedy algorithm. The greedy algorithm required 19s to run, and the genetic algorithm required 40s to run.

	Greedy Algorithm	Genetic Algorithm
Partition	P1={5,7,11} P2 = {1,2,3,4,6,8,9,10,12, 13,14}	P1={5,11} P2={1,2,3,4,6,7,8,9,10,12,1 3,14}
IDDQ (nA)	$IDDQ_1$ = 1581.60 $IDDQ_2$ = 3668.31	$IDDQ_1$ = 1466.28 $IDDQ_2$ = 3674.50
TAT (Cycles)	790	775
Run Time (s)	19	40

6.0 Conclusions

In this paper, we studied the problem of IDDQ testing of system chips implemented in deep sub-micron technologies. The leakage current in such chips is quite large, making it hard for us to apply current testing. Expensive technological solutions exist for solving the problem, such as the use of highly sensitive current sensors, the use of low temperature testing, or the use of Silicon-on-Insulator technology for implementing the chip. We propose an approach based on partitioning the set of cores and testing the partitions one at a time in order to improve the confidence of the testing procedure. For example, in the 14-core SOC example which we experimented with, the faulty IDDQ current has a Gaussian distribution with a mean of 5149.91 nA and a standard deviation of 257.5 nA. The fault-free IDDQ has a mean value of 5140.78 nA. Given that there will be variations in the measured value of fault-free IDDQ due to variations of manufacturing processes, voltage, and temperature, it becomes difficult to distinguish between faulty and fault-free chips. On the other hand, if the cores are divided into two partitions, the IDDQ for the two partitions have mean values of 1466.28 and 3674.50 respectively. This "separates" the faulty and fault-free IDDQ distributions and allows the test equipment to distinguish faulty and fault-free chips. We formulated the partitioning problem as a constrained optimization problem and proposed a greedy heuristic and a genetic algorithm for solving the problem. We presented techniques for estimating the test time and IDDQ for cores as well as the SOC. Experimental results on two synthetic benchmark circuits indicate that the genetic algorithm yields near-optimal results.

Acknowledgements: This work was carried out when the Rahul Kumar was an MS student at IIT Delhi. The authors would like to acknowledge Controlnet, India for providing the computational facility available to carry out the research reported in this paper.

References

1. Sreejit Chakravarty and Paul J. Thadikaran. **Introduction to IDDQ Testing**. Kluwer Academic Publishers, 1997.
2. A. Gattiker, P. Nigh, D. Grosch, and W. Maly. **Current Signatures for Production Testing.** Proc. of IEEE International Workshop on IDDQ Testing (IDDQ '96).
3. A.C. Miller. IDDQ Testing in Deep Submicron Integrated Circuits. Proc. of ITC, 724-729, 1999.
4. F. Najm, I. Hajj, and P. Yang. Computation of bus current variance for reliability estimation of VLSI circuits. Proc. of ICCAD, 202-205, 1989.
5. R. Rajsuman**. IDDQ Testing for CMOS VLSI.** Artech Publishing House, USA. 1995.
6. R. Rajsuman. **Design for IDDQ Testing for Embedded Cores based Sysem-on-chip**. Proceedings of IEEE International Workshop on IDDQ Testing. Pages 69-73, 1998.
7. C.P. Ravikumar, A. Verma, and G. Chandra. **A polynomial-time algorithm for power constrained testing of core based systems.** In *8th Asian Test Symp*., pages 107-112, 1999.
8. C.P.Ravikumar and V. Saxena. **TOGAPS -- A Testability Oriented Genetic Algorithm for Pipeline Synthesis**. International Journal of VLSI Design, Gordon and Breach Publishers, 5(1), 1996. Pages 77-88.
9. C.P. Ravikumar and N.S. Prasad. **Evaluating BIST architectures for low power.** In *7th Asian Test Symposium*, pages 430-434, 1998.
10. C. P. Ravikumar and A. K. Gupta. **Genetic algorithm for mapping tasks onto a reconfigurable parallel processor**. IEE Proc-Comput. Digit. Tech, 142(2):pp81--96, 1995.
11. M. Sachdev**. Deep Submicron IDDQ Testing: Issues and Solutions**. ED&T. 271-278, 1997.
12. K. Sawada and S. Kayono. An evaluation of IDDQ versus conventional testing of CMOS sea-of-gate ICs. Proc. of ITC. 158-167, 1992.
13. Y. Zorian. **Challenges in testing core-based system chips**. IEEE Communications Magazine. Vol. 37, No. 6, Pages 104-109, 1998.
14. IEEE P1500 General Working Group Website. http://grouper.ieee.org/groups/P1500/
15. Semiconductor Industry Association (SIA). **The National Technology Roadmap for Semi-conductors**. http://public.itrs.net/files/1999_SIA_Roadmap/Home.htm

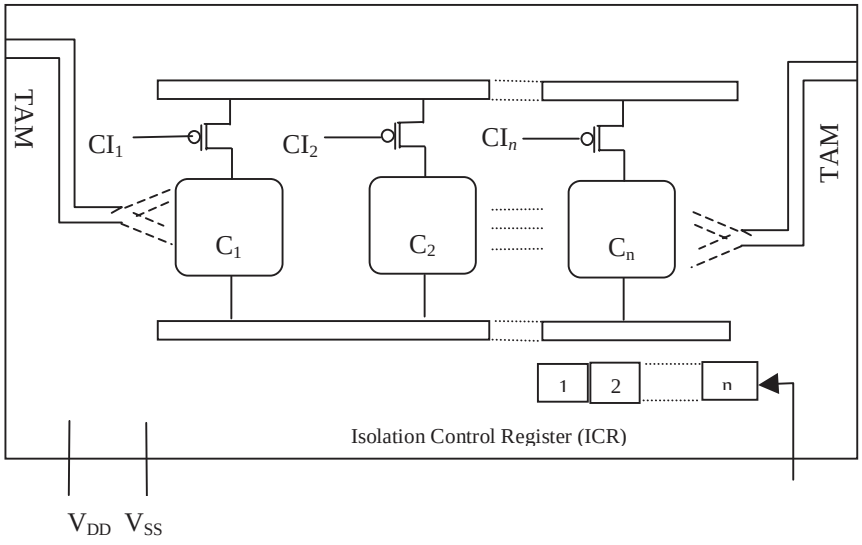

Figure 1: P1500-Compliant IDDQ

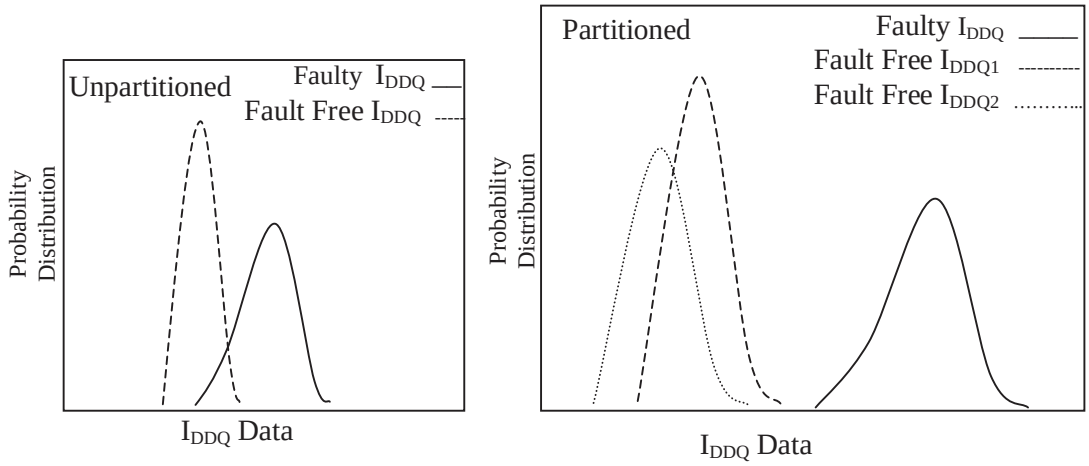

Figure 2: Illustration of Reliability of IDDQ Testing