A METHODICAL APPROACH FOR SELF CHECKING AND FAULT TOLERANT DESIGN

Y.RAJASREE AND DR.N.R.ALAMELU

Sridevi Women's Engineering College Hyderabad, INDIA

ABSTRACT

Building fault tolerance into a system cannot be an afterthought or add on approach. Design for fault tolerance needs to begin with the early stages of system conceptualization, requirement, specification and system design. The ultimate goals of a computer system affect its design philosophy and design tradeoffs. The cost of fault tolerance must be weighed against the cost of error or failure. This paper gives the design methodology for fault tolerant systems. Identify class of expected faults over the life time of the system**.**

Keywords: *Res_Code, Circuit, FPGA, Fault, System, System, Clock, Model*

1. INTRODUCTION

Experimental implementation of the design method for embedding fault tolerance capabilities into high level digital systems is demonstrated. The method starts with standardized behavioral level system descriptions and systematically transforms it into high level synthesis description of VHDL based self checking digital models. The method starts with a standardized behavioral level system description and systematically transforms it into an implementation level circuit design with fault tolerant parts built in. The transformation process aims to keep the changes made in the model transparent from the view point of the designer, in order to maintain comp ability between the original system models and to minimize the manual interaction needed to implement fault tolerance. The method is intended to be easily incorporated into existing digital system design environments.

The implantation of the fault tolerant capabilities is performed by replacing common VHDL data types with alternate, self checking capable versions. This way the initial high level model needs only a minimal modification, and maintains comp ability with high level simulation and verification tools. The code parts implementing fault tolerance are implemented as a separate VHDL package of register transfer level descriptions, and they are included into the result of high level synthesis automatically, or within minimal user intervention.

2. ERROR DETECTION ON BEHAVIOURAL LEVEL

On the behavioral; level of system modeling, the most common data types are members (usually integers, floating point numbers). Both the correctness of the result produced by the system and the control flow of the algorithms themselves depend severely on the integrity of the numeric values. However, numbers are transformed to multi bit data lines at lower levels off abstraction, so physical level faults (stuck at, bridging etc.) are represented as corrupt values at the behavioural level. Therefore the numeric values of the high level model need to be protected against unintentional value changes.

There are numerous methods for this purpose.[17]. The application of residue codes [2] have attractive advantages: relatively low calculation requirements and error detection capability. A simple residue code was implemented for integer numbers. Applying the mod 3 residue code involves the extension of all integer values by a separate residue value that contains its remainder modulo 3. Additionally, the consistency between the numeric value and the residue must be held during the operations on the integers. This task can be

solved in every elegant way in VHDL, due to certain syntactic properties of the language. As all the VHDL operators are treated basically as functions, a feature called operator overloading is provided. The code is given below

```
Package integer_rescode is
    Subtype res_code is integer range 0 to 3
; constant
Invalid integer_rc is record
    Value : integer;
Rescpde : res_code;
End record;
Function "+" (1, r : integer_rc ) return
integer_rc ; function "_" (1,r: integer_rc)
return integer_rc ;
End integer_res_code;
Package body integer_rescode is
    Function rescode ( arg : integer ) return
res code is variable result;
    Res_ code;
        Begin
            Result := arg mod 3 ;
return result;
    End res_code;
        Function "+" (1,r : integer_rc )
return integer_rc is variable result ;
        Integer_rc;
        Begin
            Result.value := 1.value +
r.value ; if (rescode (1) /= 1.rescode)
        Or (rescode (r) /= r.rescode ) then
result.rescode := invalid_rc;
        Else
        Result.rescode := result.value; end
if;
        Return result ; end "+";
        End integer_rescode;
```

Here a data type integer_rc is declared as a record ; its members are value (that carries the numeric value of an integer) and rescode (that carries the residue code).Then the function implementing the standard operators "+" , "_" , etc are defined on integer_rc type operands in a way that they return the result of the same operator on integer in value, and the residue code of this result in residue code. The constant invalid_rc is defined for error detection purposes , as it represents an illegal mod 3 remainder, rescode is set to invalid_rc if the residue code of any operands was incorrect. The illegal value of rescode is propagated in all subsequent operations, thus ensuring that as long as the effect of any value change caused by a fault is detectable by mod 3 residue codes, it will be observable on the affected outputs of the system as well.

Embedding residue code based fault tolerance into behavioural level VHDL circuit descriptions is most simple. It consists of including an alternate VHDL numeric package (that contains the declaration of the residue code protected integer_re type and the definitions of the standard VHDL operators on this type), and replacing all integer type definitions with integer_reo .

3. TRANFORMATION INTO REGISTER TRANSFER LEVEL

Modern digital circuit design systems usually offer automatic silicon layout generation from low level circuit descriptions. Therefore behavioural level models must be transformed into a lower level. In this research, the Synplicity Synplify Pro [4] system has been chosen, due to its free availability and positive past experiences.[5].

The embedded fault tolerant features of the behavioural models must also be transformed .As most high level synthesis systems transform behavioural level operations in a predefined low level functional units overload operators. Obviously, the high level synthesis tool must accept the modified integer_re type and convert it appropriately to a lower level data type. As this data type is usually the standard bit_vector or std_logic_vector type with globally defined size (which is user supplied design parameter), it can be easily extended with the two extra data lines needed by the mod 3 residue codes. In the case of Synplicity Synplify Pro , a set of FU s is supplied in low level VHDL description format, so their modification is just as simple as the modification of the behavioural model.

4. SIMULATION AND RESULTS

The sample circuit used for demonstration of the embedding of fault tolerance was the GCD example supplied with Synplicity Synplify Pro. The circuit

realises the traditional Euclidean Algorithm for calculating the greatest common divisor of two integer numbers as a sequential digital circuit.

Xilinx FPGA development tool was used to convert the RTL VHDL model into a Xilinx FPGA .The overhead in the number of equivalent gate inputs was 48% with a 8 bit wide integers. These values are significantly higher than the expected values according to [39]. The actual overhead is affected because of the following factors:

- The most optimal implementation of the residue code checker (it was necessary to re-implement it almost completely due to the different VHDL subset used by the FPGA design environment)
- The sample circuit was mostly data dominant, with a very simple control sequence.
- The FPGA synthesis tool generated the checker as a combinational circuit, optimised for speed. A slightly slower, sequential implementation would have resulted in a smaller hardware overhead.
- The target architecture applied in Synplicity Synplify Pro resulted in the gate level implementation of the residue checker in each functional unit. If the residue checker is implemented as a separate unit, and is shared between the FU s of the circuit, the overhead with respect to the whole circuit is obviously smaller.

5. COMPARING DIFFERENT FAULT MODELS

Fault injection is widely used for evaluating dependable systems. In this research, various fault models used for fault injection are compared. The experiment has been performed by using the simulation based fault injector VERIFY (VHDL based Evaluation of Reliability by Injecting Faults efficiently).

The technique implemented by VERIFY enables a completely automated evaluation of system dependability features. For starting an experiment, the user specifies the simulation time and the number of faults k which should be injected. Each experiment consists on fault free run (golden run) and k runs where for each run exactly one fault is injected.

Once the experiment has been started, the simulator injects the required number of faults without any user intervention. Each time a fault has to be injected, the simulator determines automatically the type, location, time of occurrence and duration of the fault according to the fault descriptions in the model. The dependability of the complete system can be evaluated by injecting several thousands of faults within a simulation time which is representative for the service the system has to provide.

During the fault injection experiments, a trace of all signal values is logged for the golden run and fpr the time after a fault has been injected. In order to speed up the simulation time, the experiments are carried out by a technique called multithreaded fault injection described by Guthoff and Siehin

6. EXPERIMENTAL STUDY

In order to compare the fault models, an experiment using VHDL model of the DP-32 processor is set up. In the following two sub sections, a summary of the basic characteristics of the DP-32 processor and the fault model for which fault injection experiments have been performed are presented. A detailed description of DP -32 processor model can be found in [8].

7. THE DP32 PROCESSOR

The DP 32 is a simple 32 bit RISC processor which has been chosen to compare the results with the fault injection experiments .A VHDL model of the processor presented at RTL level and has been modified so that Synopsys Synthesis tool automatically generates a gate level VHDL model.

The A/D bus is used to fetch instructions and to transfer data between registers and memory. The actions of the DP-32 are controlled by a single finite state machine (FSM). The model used does not support instructions for multiplication and division. The register file of the original model supports 256 register has been reduced to 8 registers.

The fault injection experiments are performed with two different gate level models of the DP-32. The models are automatically generated by the synthesis tool

using different cell libraries. The library for the simple model includes only the fundamental gates: an AND and OR gate with two inputs : inverter, tristate driver, D flipflop and D latch. Due to the simple library the generated gate level model of the DP-32 includes 32.4 5 more gates than the advanced gate level model generated with an advanced library. The advanced library does not include the additional cells for the NAND gates, NOR gates, XOR gates, Multiplexes and Full adder. The logic gates are available in versions with 2, 3 and 4 bit inputs.

During the simulation of both gate level models the test program given in Figure 9.1 is processed . First, it resets register r0 to zero. Then, it increments r2 starting at 0 until r2 is equal to 10, where it restarts at r2 equal to zero. The simulation of one cycle (counting from 0 to 100) needs 6 µs at 20ns clock cycle length.

```
                    Initr0
Start :                     addq(r2,r0,0)
    ; r2:=0
Loop :                      sta(r2,
counter)            ;counter :=r2
                    Addq(r2, r2, 1)
    ; increment r2
                    Subq (r1, r2,10)
    ;if r2=10 then
                    Brzq(start)
    ;restart
                    Braq(loop)
    ;else next loop
Counter :           data(0)
```

Test Program

8. FAULT MODELS

Three different fault models have been chosen the well known stuck at x fault model at gate level, bit flips in registers of the register transfer level and stuck at x at pin level of the processor.

- The stuck at x model at gate level is widely used in conjunction with the test pattern generation .The customary approach of allowing stuck at 0 or stuck at 1 only at output signals of the components by adding the same possibility of fault for input signals. If the output of a gate drives one signal which will be used as an input for several other gates, allowing faults only at the output of the gate would always affect all components connected with the affected signal. This fault model has been presented which allows the extension of a NOT gate.
- For the second fault model, the single bit flip model i the internal registers and latches of the DP-32 processor has been chosen. This model is used by nearly all tools which are based o the approach of software implemented fault injection .For the experiments the faults have been injected uniformly distributed over all bits of internal registers including the bits of the finite state machine of the processor.
- The third fault model chosen is the injection of faults at the pin level of the DP-32 processor. Like in the gate level stuck at fault model, which is the most detailed one, single stuck at faults uniformly distributed over all has been chosen. For this purpose the test bed for the experiments with a socket which connects the DP32 processor with the main memory via the address and data bus and its control lines was extended. The location of the faults to inject is restricted to the socket and, therefore, to the pins of the processor. With this fault model, all signals going to or from the processor can be corrupted.

In all experiments the same frequency of occurrence and mean duration time is used. It would be easy to adjust these values if more realistic rates or duration times are known. Four experiments have been performed in order to evaluate the influence of different parameters on the results of the fault injection.

- In the first model the simple gate level model of the DP32 was used and only internal stuck at faults were injected at the input and output ports of each gate.
- The evaluation of the influence of different realisations on the behaviour of the system after fault injection were measured by comparing simple with the second experiment. For this purpose the advanced level gate model is used. The fault model which has been used was the same as in the simple experiment i.e stuck at faults at the input and output of every gate inside the processor.

- For the third experiment, the advanced gate level model DP32 was used in conjunction with the bit flip faults at all internal registers. The advanced gate level model of the processor has been favored over the simple because of a shorter simulation time due to fewer gates.
- In the last experiment stuck at faults were injected into the socket of the processor. As no faults have been injected inside the DP32 , any level of the description of the DP32 could be used as a model of the CPU.

Every experiment consisted of 1000 different test runs; where for each run one single fault has been injected. The behavior of the system was observed for 2µs after injecting the faults. The mean duration of the fault was observed to be 20 ns. All experiments have in common that no faults have been injected into the clock, reset and the RAM components of the test be observed.

9. RESULTS AND DISCUSSION

The description and specifications of the analog ore used in mixed signal circuits is presented. Three mixed signal chips are used.

Three digital soc test benches have been used. Five analog cores have been added to the cores. These shall be referred as m1, m2 and m3 respectively. The analog core consist of a pair of base band transmit path with a bandwidth of 50 KHz. a base band down conversion path and a general purpose amplifier.

These analog cores are taken from a commercial base band cellular phone chip. The test specifications is given in Table 9.1
For the I-Q transmit path pair, six distinct specifications based tests are defined. These include the pass band gain, the cut off frequency, the attenuation levels at 1 and 2 MHz , the third order input intercept.

10. COMPARISON OF FAULT MODELS

The graphs indicate how the circuit reacts to the faults injected. The graph G(t) indicates the percentage of injected faults which caused faulty states in the circuit at a given time t after the fault has been deactivated. The function 1-G(t) is the recovery time distribution .In addition to the recovery time distribution , the diagrams show the probability that the system is not able to recover from an injected fault within the observable interval T. This probability is G(t). Each diagram on the right side is a manifestation of the first nano seconds of the graph on the left side.

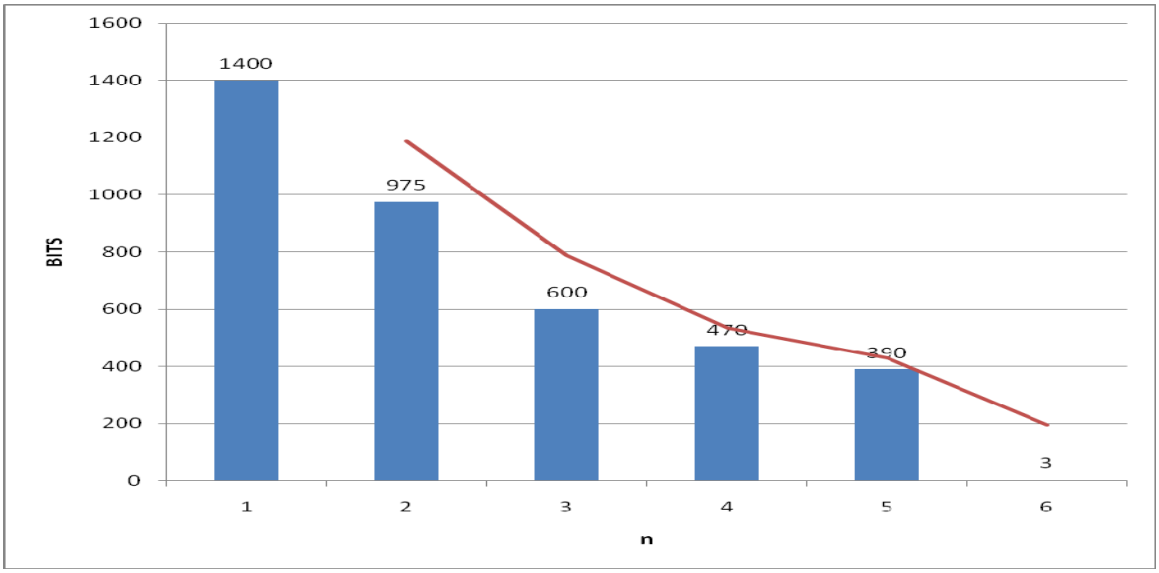

Figure 1 : RBF coverage and logic size as function of n(1%Xs)

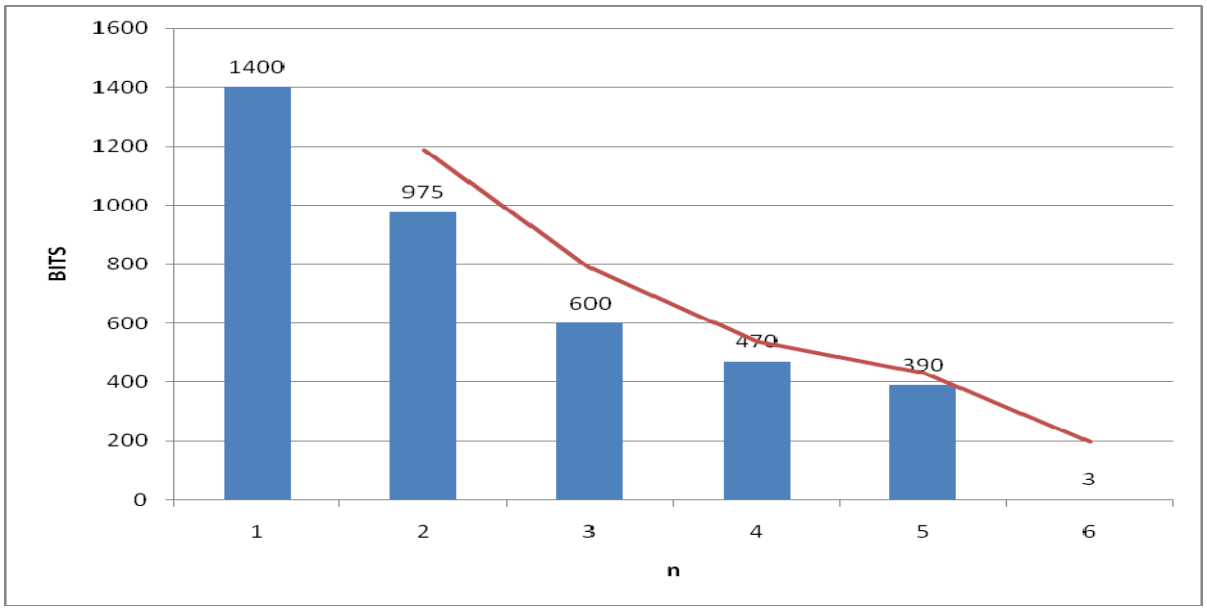

Figure 2: Number of masked bits and logic size as function of m (3% Xs)

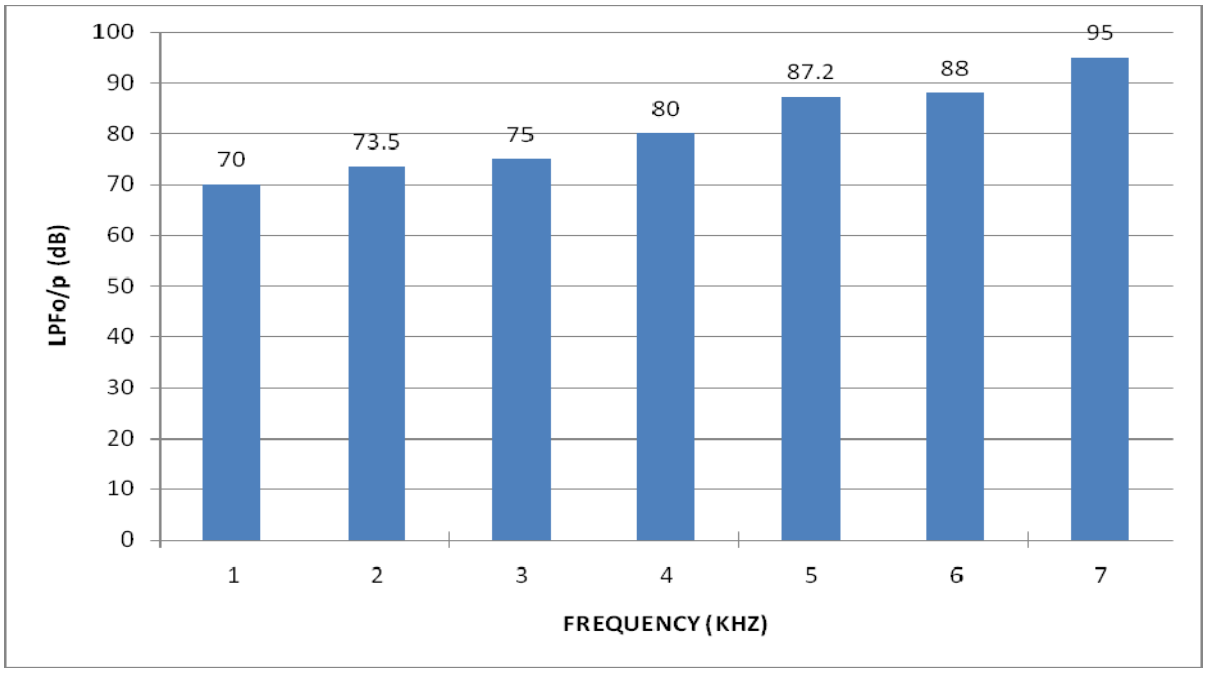

Figure 3: RBF coverage and logic size as function of n (3%Xs)

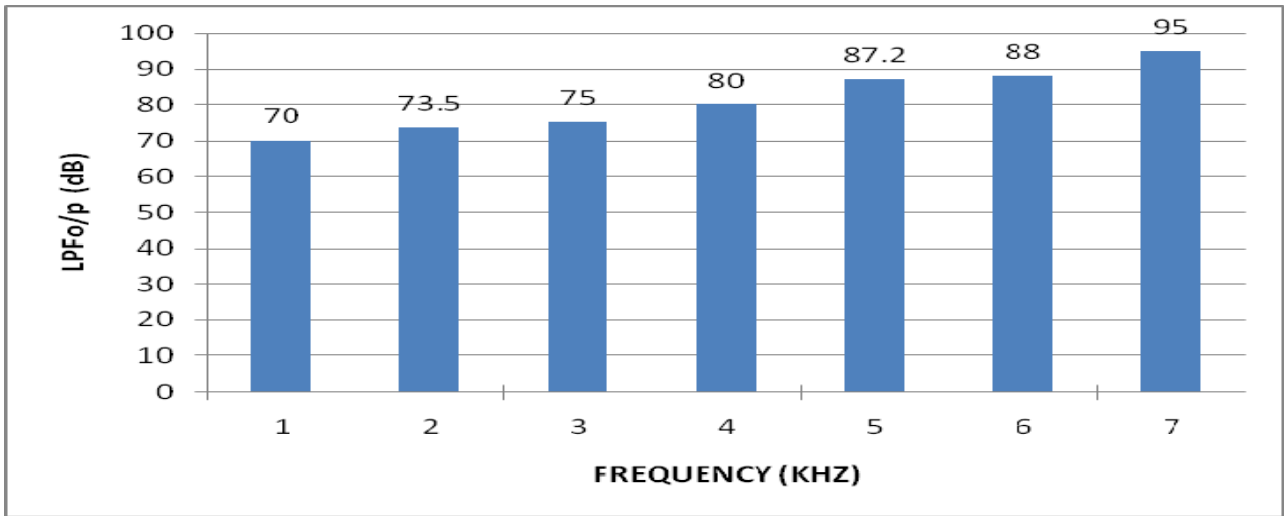

Figure 4: Frequency spectrum of applied test.

It is observed that the reaction of the system heavily depends on the type of the injected fault and on the used gate level model e.g. faulty states induced by internal stuck at faults and external stuck at faults.

The results demonstrate that faults have to be injected at least at gate level in order to represent the correct timing behavior of the digital system after the fault has been injected and help the designer of a dependable system to find the weak components.

11. CONCLUSION

The results summarized above indicate that reliability analysis can indeed be formalized so as to reflect the ability of the user to rely on the computations a computer performs (as opposed to the computer itself). It is also clear that the research performed to date is only a beginning in this direction, and there

are a number of topics that deserve immediate investigation. The first of these is the further exploration of examples that illustrate-how the probability of computation-based) success can be evaluated , Secondly, other reliability measures such as "meantime- to-failure," "fault-tolerance," "availability" and "recoverability" should be given a computation-based formulation. In particular, recoverability (i.e.,"coverage" **)** should be focused on since it is inherently a computation-based measure. Third, there is need to consider specific instances of the general model that are closer to particular applications problems. The purpose of the general model has been to formalize reliability measures along with the information required to evaluate the measures. However, to obtain a useful tool for assessing the reliability of a special class of systems (e.g., aircraft computers), the model must be specialized to permit practical methods of evaluation using practically available data.

Fourth, the evaluation methods just referred to must be investigated. This includes simulation methods as well as analytic methods. When simulation methods are employed, the simulation should account not only for the probabilistic nature of faults but also for the probabilistic nature of the computational environment. The end product of this last investigation should be a set of algorithms which, given reasonable constraints on computer time, computer memory and cost, can evaluate a set of computation-based reliability measures

REFERENCES

[1]. A.Cron ,“ IEEE P1149.4-Almost a standard ” ,in Proc. IEEE Int. Test Conf. (ITC) ,1997, pp. 174-182

[2]. Bavuso .S.J (1984), “An user Guide of CARE III ’’, IEEE Proceedings Annual Reliability and Maintainability Symposium , pp. 382-389.

[3]. Barbe ,D.F (1983),Very Large Scale Integration-Fundamentals and Applications, Springer-Verlag, European Simulation , pp. 365-369.

[4]. A.Lu and G.W.Roberts. (1994) , “ An oversampling based analog multi tone signal generator ’’ , IEEE Trans. Circuits Syst. II, Exp. Briefs , vol. 45, no. 3, pp. 391-39.

[5]. A.Lu, G.W.Roberts and D.J.Johns. (1994), “A high quality analog oscillator using oversampling D/A conversion techniques ’’, IEEE Trans. Circuits Syst. II Exp. Briefs, vol 41, no. 7, pp. 437-444

[6]. B.Koupal, T.Lee and B.Gravens, “Bluetooth single chip S---------- radios: Holy grail or white elephant [online].Available : http:// www.signiatech.com/pdf/paper_two_chip.pdf

[7]. B.Dufort and G.W.Roberts.(1999), “ On chip analog signal generation for mixed signal built in self test ’’, IEEE J. Solid State Circuits, vol. 45, no. 3, pp. 391-394.

[8]. 8.C.Metra, M.Favalli and B.Ricco.(1999) “ On line Detection of Logic Errors due to cross talk, Delay and Transient Faults ”, Proceedings of ITC , pp. 524-533.

[9]. 9.C.S.Wallace. (1964), “A Suggestion for a Fast Multipliers”, IEEE Transactions on Electronic Computers, pp. 14-17.

[10]. 10.C.Su and Y.T.Chen . (2000), “Intrinisic response extraction for the removal of the parasitic effects in analog test buses ’’, vol. 19, no. 4, pp. 437-445.

[11]. 11. C.Y.Pan and K.T.Cheng. (1997) ,“Pseudorandom testing for mixed signal circuits ’’ , IEEE Trans. Compt. Aided Des. Integrated. Circuits Systems, vol.16, no. 10, pp. 1173-1189.

[12]. 12.D.A.Johns and K.Martin. (1997) , Analog Integrated Circuit Design , New York; Wiley.

[13]. 13.D.Gizopoulos, A.Paschalis and Y.Zorian. (1999), “An Effective Built in Self Test Scheme for Array Multipliers”, IEEE Transactions on Computers, vol. 48, pp. 936-950.

[14]. 14. D.K.Pradhan, ed al. (1986), Fault Tolerant Computer System Design, Prentice Hall, Englewood , New Jersey.