Testing Byzantine Faults

Bhatiavva B. Bhadviya
Department of Computer Science and Engineering,
University of Michigan, Ann Arbor, USA-48109
bhavi@umich.edu

Abstract

Byzantine fault is a cumulative term given to the set of both permanent and transient faults caused in circuits which effectively present themselves as different errors at different nodes in the same circuit. Such faults have become significant with the shrinkage in node (process) technology with process variations, radiation strikes and quantum-mechanical effects being majorly responsible for causing such faults. Such faults not only vary with time but also cause different effects (errors) every time they occur. In this survey we will try to recognize such faults at the circuit level, identify their exact cause and also enumerate their consequences. The survey will also discuss the most efficient byzantine fault model and shares light on probable rectification or preventive scenarios for such faults. We will specifically look into the open-segment byzantine fault model intended for byzantine faults caused due to open interconnects as well evaluate the byzantine faults caused due to the resistive bridging faults and support the corresponding IDDQ test procedures with experimental data.

Index Terms—Byzantine faults, open segment faults, resistive bridging faults, symbolic inject and evaluate paradigm
I. INTRODUCTION

Identifying manufacturing and design faults have been one of the most critical steps in increasing the production yield of ICs. Being able to pin-point defects in the IC during the phase of early prototyping helps prevent the same errors from occurring again. Test engineers perform fault diagnosis and try to resolve errors up to the level of a specific transistor or an interconnect. Stuck-at-fault model has been very successful in defining and identifying all types of permanent faults.

But with increasing nature of uncertainty in IC manufacturing process as well as shrinkage in the process technology we are observing a new family of transient faults. These transient faults not only happen at variant time spans but also cause varying effects every time they occur. All different faults require different fault diagnosis strategies. Further Fig 1 shows the classification [1] of major fault types in a logic circuit.

There is a possibility that the faults occurring are single faults or multiple. Further Single faults can again be the effect of one net or be the effect of two nets. S-a-0 and s-a-1 are the most common example of single net faults while resistive bridging faults is the most prevalent example of the two net faults. Node faults are the names given to the faults caused due to the malfunction of a gate/transistor thereby changing its functionality. Open faults are the faults caused due to stuck-at-open conditions of interconnects. Logical fault is the term assigned to the logical error specific to a particular node. In this survey we will concentrate our efforts on Byzantine faults. Byzantine faults are one and two net faults that show ambiguity. By definition[1] a fault site S is a byzantine fault if and only if there exists at least two logical gates A and B connected to S and that A interprets the value of S as ‘1’ and B interprets its value as ‘0’.

Until now for fault analysis we assumed that a particular fault creates a logical error right at the fault site. But this may not be true for a byzantine fault, and absence of such an assumption results in exponential complexity of fault analysis. We will try to address this complexity by touching upon the new fault models designed specifically to account for byzantine faults.

Earlier stuck-at model was used to define such faults and then explicit enumeration was used to diagnose byzantine faults. But both these methods were exponential as for each fan-out site with k branches there was a possibility of $2^k$ ambiguous logical values. To emphasize this complexity lets look at the table1 which enlists the degree of maximum fan out branches in some ISCAS85...
circuits.

Table 1: Degree of Maximum Fanout Branches

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Nodes</th>
<th>Maximum Fanout Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>175</td>
<td>28</td>
</tr>
<tr>
<td>C1908</td>
<td>465</td>
<td>25</td>
</tr>
<tr>
<td>C2670</td>
<td>694</td>
<td>56</td>
</tr>
<tr>
<td>C6288</td>
<td>2353</td>
<td>3</td>
</tr>
</tbody>
</table>

As we see that with such high complexity at our hand it is wise to look for better fault models and fault detection algorithms when it comes to Byzantine faults. In this survey we will partition our efforts into identifying the sources of all such Byzantine faults and then discuss a few models specific to stuck-at-open and resistive bridging faults. The last section will also look into specific byzantine examples and their rectification.

**II. SOURCES OF BYZANTINE FAULTS**

Byzantine faults occur more frequently that what a designer may think. In fact the frequency is about $10^{-9}$ faults per operational hour [4]. A most common example of a byzantine fault is the stuck at ‘1/2’. Let us consider the characteristic transfer function of a gate with a $V_{dd} = 1.2V$ as shown in Fig 2.

We can see that depending upon the threshold of the next gate, the output may belong to any of the three regions. In general digital electronics only considered 0 and 1 as the two possible outcomes. But with size shrinkage and increasing process, voltage and temperature (PVT) variations the output can very well occur in region2.

In this scenario this intermediate voltage can be interpreted by the destination gates as either 1 or a 0 (A classical case of Byzantine fault).

Such intermediate logic values can also appear due to an open segment as shown in Fig 3. Due to the disconnection between the fan out branches, the destination gate can interpret the input depending upon its own structure and threshold.

Such threshold dependent signal ambiguities are mainly caused due to PVT variations across the die. But apart from this alpha particle strike or
cosmic rays also interfere with the original logic and may force a ‘1’ appear a ‘0’ for time being causing an effect similar to the byzantine fault.

Figure 4: Byzantine phenomenon due to alpha particle strike or node error due to PVT variations

Another major site for such faults is the output of a flip flop. A metastable flip flop will generally lie within the region 2 (Fig 2) of operation and hence with drive all its fan-outs to the value ‘½’, thereby being a principle candidate site for a Byzantine fault.

III. CONSEQUENCES OF BYZANTINE FAULTS

Byzantine faults can have very devastating effects on the system. The ambiguous voltage level created at the fault site can propagate through various logic levels an even cross the fault-containment zone thereby making the entire system unusable.

Consider a data transfer example of a “Schrödinger’s CRC” [4]. Here say A and B are the two destinations and let RxA and RxB be the data received after transmission side encountered on stuck at ½ logic level. Since the CRC calculation is a linear XOR of the data stream all the CRC bits are affected and this results in completely ambiguous data transfer.

Byzantine faults also cause some serious effects in the time domain. They can cause a data edge to delay enough so that it coincides with the clock edge causing metastability issues which are further amplified in the next clock cycles as this metastable value travels throughout the next stage of the circuit causing more byzantine issues.

The probability of Byzantine faults crossing a fault containment zone is about $10^{-5}$[4] per hour of operation and if we have four such containment zones in our circuit, this evaluates to the about $4*10^{-5}$ failures per hour due to byzantine fault alone. This value definitely exceeds the limits of any dependable system and hence there is an utmost need to address this growing probability of byzantine faults in the digital circuits.

IV. BYZANTINE FAULT MODELS

Although until now the probability of a byzantine fault occurring were very low, there was not much concern in the IC design and manufacturing industry to actually spend time and effort in mapping byzantine generals problem with the mass productions ICs. Nevertheless there are 3 byzantine fault models in literature which we will discuss in this section.

A. Symbolic Inject and Evaluate model for open segment Byzantine faults

This is one of the most recent byzantine fault models [1] and aims to pin-point the byzantine
fault site amongst all the fan-out sites in the circuit under diagnosis (CUD). It assumes that the design is implemented with the full-scan design methodology and that the complete functionality can be defined as a combinational circuit. For sequential circuits, the flip flop boundary can be used to separate each stage into a purely combinational functional definition.

It takes into consideration the Primary outputs (POs) of the manufactured chip and the equivalent gate level netlist. In case of byzantine fault occurring in the IC, the POs of the actual fabricated CUD will show mismatch from the simulated gate level netlist.

Fault free logic simulation simply evaluates the signal values of each net and the primary output for all the input vectors.

Symbolic Injection is a procedure in which each fan out branch (one at a time) is disconnected from its parent and then reassigned a Boolean variable instead of a Boolean value. The primary outputs are then recalculated in terms of those Boolean variables using symbolic simulation or event driven simulation. Fig 6 depicts the intermediate state of the CUD during the symbolic injection phase.

Symbolic propagation involves propagating these Boolean variables up to the primary outputs. This can be done using a standard symbolic simulator that uses BDDs (binary Decision Diagrams).

As shown in Fig 5 the mismatched outputs are the 1\textsuperscript{st} and the 2\textsuperscript{nd} bit from the top. The corresponding input vector that causes the primary outputs to mismatch is called the Failing input vector. The entire procedure to find the byzantine fault is an effort to resolve this mismatched output. This method can be summarized as:

1) Fault free logic simulation
2) Symbolic Injection
3) Symbolic propagation
4) Curability checking
The propagated values at the output are called as the “React function”. Once we obtain the react function we perform the curability check.

Say a the i\textsuperscript{th} output bit (O\textsubscript{i}) suffered a mismatch and that a signal ‘a’ with k fanouts (a\textsubscript{1},a\textsubscript{2},...,a\textsubscript{k}) exists in the CUD. The output O\textsubscript{i} is said be a byzantine curable output if and only if the mismatch at i\textsuperscript{th} bit of output can be resolved by assigning (Injecting) specific values to a\textsubscript{1}, a\textsubscript{2}, ..., a\textsubscript{k}. Similarly the output O is said be a byzantine curable vector if and only if all the mismatched output bits can be resolved by assigning (Injecting) specific values to a\textsubscript{1},a\textsubscript{2},…,a\textsubscript{k}.

We continue this procedure until all the fan-outs are checked and symbolically simulated. And every time we find a curable vector for a given fault we assign that particular branch as a likely candidate for the byzantine fault. We calculate the first hit index of such fault sites giving us a performance metric of this approach.

Another interesting aspect of this approach is that it can be used over the layout segment model of the circuit. The layout interconnect can be modelled as a node of interconnects and each of them forming the inject points. The similar procedure can then be used to find byzantine fault sites.

B. IDDQ test procedure for Byzantine faults caused due to bridging faults

Such type of faults are caused due to the unintentional shorting of two or more interconnects or transistor terminals. Consider the fact that the source and drain of a transistor are shorted. This means that irrespective of the gate controlling voltage this particular transistor will always remain ON. This results in excessive leakage in the circuit and can be used to monitor such faults.

Consider an XOR circuit as in Fig 8 without any bridging fault across M1. The current flowing through the XOR branch is nominal in a few nA.

Now consider the same circuit with the source and drain of M1 shorted. This short models the bridging fault across M1 as well as the node error (logical error) as in Fig 9. In this case we observe that current leakage has increased several times (400,000 times). This increase in leakage can be used to monitor the power and identify the region of byzantine fault existence. According to an
intuitive understanding, the entire chip region can be subdivided into a number of equal regions and separate power monitoring can be provided for each region.

Figure 9: Faulty XOR circuit, M1’s Drain and Source are shorted, Current = 425.2uA

The increase of about 400K times in current consumption for a different input sequence is greatly higher than caused due to process variations and hence can be used to accurately monitor power and hence the location of such a byzantine faults.

V. BYZANTINE FAULT RESISTIVE DESIGN

The most important aspect of byzantine fault resistive design is to prevent process variations during manufacturing at the most keep them at minimum. The designers can take more care in sizing and placement of components and the interconnect. Although these measures will benefit us a little but definitely does not guarantee a byzantine fault free design.

The most intuitive solutions would be to use a voting system for all the major signals in the design. A simple 3 voter system is shown in Fig 10.

Table 2 gives the truth table for the above 3 voter system. From the table we conclude that as long as two of the 3 inputs are same we can be sure of the output being free from any byzantine effects.

Table 2: Truth Table for a 3 voter system

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1/2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1/2</td>
<td>1</td>
<td>1/2</td>
</tr>
</tbody>
</table>

In general we would require about $2f+1$ redundant systems or voting elements for making our system resistive for $f$ byzantine faults. This type of byzantine fault resistive system can be used in sensor systems that need very high dependability as in a nuclear reactor setup or even for aircraft flight data collection. People have tried Quad/Triple
module redundancy but once a byzantine fault occurs, such configurations are easily compromised.

A more preferred design would be to have the primary output at every fan out branch to be encoded in an unordered code and to make sure that the total inversion parity from all the fan-outs to the primary output is the same.

Yet another explored area of single byzantine fault resistant design is to have self checking circuits which calculate the hamming distance between each of the inputs and use a CRC like XOR mechanism to generate an error free output. The above mentioned correcting mechanisms have really negative impacts when it comes to area. In addition these concepts cannot be extended to the complex and million transistor design of today. These error correcting circuits actually increase the critical delay of the circuit and hence are not a viable option for mass-scale consumer production.

VI. CONCLUSIONS
Byzantine faults are indeed at an increase and will continue to do so as we continue in the sub-nanometer regime of digital circuits. Another reason for increased occurrence of these faults would be the increasing rate of wear out of digital circuits. Such transient faults cannot be modeled accurately as a stuck-at model but require special and more efficient ways to model and pinpoint the fault site. Bridging fault a class of Byzantine fault can be pinpointed by measuring physical parameters like the current and the power and hence would require to design such monitoring blocks for all our chips. Moreover there is an utmost need to design byzantine fault resistive systems.

An interesting future work would be to look into memory based system design, ie. At all fan-out sites we used SRAM arrays that have ECC error correcting codes thereby making the entire system a huge block of memory and some intermediate logic. We can use check pointing mechanisms for the entire system to guarantee the fault free operation. There are also various other frontiers to explore and we may also consider other nano-mechanical structures for obtaining greater reliability for our future system.

REFERENCES