Fault Tolerant Design of Digital Systems

Namita Arya¹, Amit Prakash Singh²
University School of Information & Communication Technology, Guru Gobind Singh Indraprastha University, New Delhi
nammoarya32@gmail.com¹, amit@ipu.ac.in²

Keywords: Static Redundancy, Dynamic Redundancy, Triple modular Redundancy

Abstract: Fault-tolerant design concepts, such as self purging, silt-out modular redundancy, overlapping parity, and cyclic duplication coding, are presented. In most of these methods, fault location can be identified and even possibly be isolated or corrected. Practical case studies involving these methodologies are presented. In addition, methodologies to test the voter circuit for possible malfunction or fault condition are given. Such methods can fault isolate the critical points such as the voters and data routers, and thereby improve the system reliability[1]. With the increasing use of computing systems in such crucial areas as medicine and space, there has come a great need for computers that remain operational in spite of hardware failures. This paper provides a brief overview of several approaches to fault-tolerant computing. Five hardware redundancy techniques are reviewed: static, dynamic, hybrid, self-purging and the reconfiguration scheme.

INTRODUCTION

Today, when computers are used in critical space missions, millions of miles from their human operators, and in biomedical systems where a human life depends on their correct operation, even a small computing error could result in the loss of a life or millions of dollars of equipment and years of research. Under such conditions, the design of computing systems which can operate correctly in spite of hardware or software failures is important. Such systems are called fault-tolerant systems. Specifically, fault-tolerant computing has been defined as the ability to execute specified algorithms correctly regardless of hardware and/or software failures[2]. The first step towards a fault-tolerant system is to build as much fault-tolerance into the system as possible[3]. Fault in tolerance is the procedure whereby the reliability of the system is increased by avoiding the causes of system failures. This is achieved before the final system is constructed, in the design phase.

A. THE IMPORTANCE OF FAULT TOLERANCE

Fault-tolerant computing is the art and science of building computing systems that continue to operate satisfactorily in the presence of faults. A fault-tolerant system may be able to tolerate one or more fault-types[9]. Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively designed system in which even a small failure can cause total breakdown. Fault tolerance is particularly sought after in high-availability or life-critical systems. The ability of maintaining functionality when portions of a system break down is referred to as graceful degradation.

Fault Tolerant design can provide dramatic improvements in system availability and lead to a substantial reduction in maintenance costs as a consequence of fewer system failures.

B. BASIC CONCEPTS OF FAULT TOLERANCE

The most widely accepted definition of a fault-tolerant computing system is that it is a system which has the built-in capability (without external assistance) to preserve the continued correct execution of its programs and input/output (I/O) functions in the presence of a certain set of operational faults. An operational fault is an unspecified (failure-induced) change in the value of one or more logic variables in the hardware of the system[5]. It is the immediate consequence of a physical failure event. The event may be a permanent component failure, a temporary or intermittent component malfunction, or externally originating interference with the operation of the system. “Correct execution” means that the programs, the data, and the results do not contain errors, and that the execution time does not exceed a specified limit.

An error is the symptom of a fault, i.e., it is the change in an instruction or a data word which is caused by the presence of a fault at the point of the physical failure event.
1. LITERATURE REVIEW

In the past, different approaches have been proposed for design verification against soft errors. These approaches can be divided in two kinds: fault injection simulation and formal verification.

Fault injection simulators run a given testbench on the design under test (DUT), flipping either randomly or specifically targeted bits. The outputs of the DUT are then compared with a golden model running the same testbench, and discrepancies are reported. Fault injection simulators come in two different flavors: on the one side there are software-based simulators like MEFISTO-L (Boué, Pétillon & Crouzet, 1998) or SST (Maestro, 2006) (which is based on Modelsim), that allow full observability and control of the simulated netlist. These tools are marred by extremely slow low-level simulation, requiring hours or days of simulation, making them unsuitable for full coverage tests[10]. On the other hand, some tools use special hardware to speed up the simulation cycle, such as FT-Unshades (Aguirre et al., 2005), which uses partial reconfiguration of an FPGA to quickly introduce bit-flips (simulating SEUs) without requiring modifications of the DUT. Although this provides a consistent speedup compared to the software based approach, it is still unfeasible to run exhaustive verification of a typical ASIC design in full, which would require the injection of bit flips in all possible Flip-Flops (FFs) at any possible time during the simulation. It is also worth noting that the results of these approaches and how they can be interpreted strongly depend on the testbench used.

Formal verification against soft-errors was introduced by (Seshia, Li & Mitra, 2007): the idea is to merge a formal model of the DUT with a soft error model, proving a given set of properties on the merged model. This requires a formal model of the DUT and a complete and exhaustive set of formally defined properties to be proven. In other words, the main issue of this formal approach is that the coverage is as good as the definition of such properties.

This work tries to overcome these limitations and provide full verification of a TMR-based DUT with reasonable analysis time. The idea presented in this paper can be classified as a fault-injection simulation, but follows a different approach as compared to previous work: instead of trying to simulate the whole circuit at once and doing a timing accurate simulation, we focus on a behavioral, timeless, simulation of small submodules, extracted by automatic analysis of the DUT internal structure, with the specific goal of detecting any triplicated FF that is susceptible to the propagation of SEUs in the DUT.

2. PROPOSED APPROACH

The starting point of our analysis is a radiation hardened circuit, protected by triplication of storage elements and voting (TMR in Carmichael (2006)). Our objective is to verify that indeed all FFs are adequately protected, and no issues were introduced, for example, by synthesis or routing tools.

Starting from a given design with \( n \) FFs, a naive testing approach for SEU-sensitive FFs would require injecting faults in all \( 2^n \) possible FF configurations, for all of the \( m \) time instants of a given testbench. This would lead to an impractically long simulation time, as typical systems consist of several thousand FFs. Our approach performs a behavioral fault injection, splitting the whole system into smaller submodules, that can be analyzed independently, allowing full verification to be carried out in a reasonable timeframe.

3. STATIC REDUNDANCY

- Also known as “masking redundancy”
- Two major techniques employed:
  - a. Triple modular redundancy
  - b. Use of error correcting codes

a. TRIPLE MODULAR REDUNDANCY

At high altitude or in space, without the protection of the earth’s magnetic field and atmosphere, integrated circuits are exposed to radiation and heavy ion impacts that can disrupt the circuits’ behavior. This paper focuses on Single-Event-Upsets (SEUs), or soft errors, usually caused by the transit of a single high-energy particle through the circuit. In particular, we consider single bit flips in memory elements embedded in logic, implemented as flip-flops. Protection against SEUs can be obtained in several ways, and in particular this work considers the protection strategy based on the triplication of the storage elements of a circuit, combined with majority voting (Carmichael, 2006), usually referred to as Triple Modular Redundancy (TMR)

MR can be either implemented during high level design (Habinc, 2002) or at a later stage by automatic netlist modification. Typically, after a new radiation-tolerant ASIC is produced, it undergoes a strict test campaign, including costly and time consuming radiation tests using particle accelerators[15]. When a problem linked to the radiation effects protection logic arises during a
radiation test campaign, it is already too late; the first prototype ASICs have been manufactured and the whole fabrication process needs to be rerun. Detecting this kind of problems before fabrication is key, therefore several software (Kanawati & Abraham, 1995; Boué, Pétillon & Crouzet, 1998; Maestro, 2006; Goswami, Iyer & Young, 1997) and hardware-based (Aguirre et al., 2005) tools for fault injection and protection verification have been proposed in the recent past. However, such tools usually are not designed to provide full SEU protection verification, and require extremely long simulation and/or execution times when attempting comprehensive fault injection and analysis campaigns.

To the best of the author’s knowledge, no commercial or academic tool providing TMR implementation verification is currently available.

For the TMR case \( N=3 \) and \( n=1 \)

\[
R_{TMR} = R_M + 3R_M^2 (1-R_M)
\]

\[
= 3R_M^2 - 2R_M^3
\]

**Note:** Another way to calculate \( R_{TMR} \)

\[
\binom{n}{r} = \frac{n!}{r!(n-r)!}
\]

\[
\begin{align*}
\binom{n}{0} &= 1 \\
\binom{n}{1} &= n
\end{align*}
\]

\[
R_{TMR} = \text{Probability of all three modules functioning} \\
+ \text{Probability of any two modules functioning}
\]

\[
= R_M + 3R_M^2 (1-R_M)
\]

\[
= 3R_M^2 - 2R_M^3
\]

**Exercise:** Evaluate \( R_{TMR} \) if \( R_M = 0.6 \) and 0.5 and 0.4

Reliability & MTBF & Failure rate

For a constant failure rate \( \lambda \),

\[
R_M = e^{-\lambda t}
\]

\[
\text{MTBF} = \frac{1}{\lambda}
\]

\[
\int_0^{\infty} R_M dt = \int_0^{\infty} e^{-\lambda t} dt = \frac{1}{\lambda} e^{-\lambda t} \bigg|_0^\infty = 0 + \frac{1}{\lambda} = \frac{1}{\lambda}
\]

Thus, for TMR where \( R_{TMR} = 3e^{-2\lambda t} - 2e^{-3\lambda t} \)
The reliability of the voter element

If the voter has the reliability \( e^{-\lambda t} \), then the reliability of the TMR becomes:

\[
R_{\text{TMR}} = e^{-\lambda t} (3e^{-2\lambda t} - 2e^{-3\lambda t})
\]

If \( \lambda_v \gg \lambda \), the reliability of the system is less than that of the original system for all \( t \).

Thus, we have to improve the reliability of the voter.

\[
\text{NOTE} : \quad \frac{5}{6\lambda} < \frac{1}{\lambda}
\]

**Figure 2:** Graph of a function \( R_{\text{nn}} \) and \( R_M \)

We should look for a more useful parameter than \( \text{MTBF} \).

**Other Parameters for evaluating system reliability**

- Reliability Improvement Factor (RIF) = \( \frac{1 - R_N}{1 - R_R} \)
- Where, \( 1-R_N \): probability of failure of non-redundant system.
- \( 1-R_R \): probability of failure of redundant system.
- Mission Time Improvement Factor (MTIF) = \( \frac{T_R}{T_N} \) at \( R_f \)
- Where \( R_f \) is some predetermined reliability (e.g. 0.99 or 0.90), while \( T_R \) and \( T_N \) are times at which the system reliability \( R_g(t) \) and \( R_N(t) \), respectively, fall to the value \( R_f \).

\[
MTBF_{\text{TMR}} = \frac{3}{\lambda} \int_0^\infty 3e^{-2\lambda t} - 2e^{-3\lambda t} dt
\]

\[
= -\frac{3}{2\lambda} e^{-2\lambda t} + \frac{2}{3\lambda} e^{-3\lambda t} \bigg|_0^\infty
\]

\[
= \frac{3}{2\lambda} - \frac{2}{3\lambda}
\]

\[
= \frac{9 - 4}{6\lambda}
\]

\[
= \frac{5}{6\lambda}
\]

**THE MAJOR ADVANTAGES OF TMR SCHEME**

Major advantages of the TMR are:

1. The fault-masking action occurs immediately; both temporary and permanent faults are masked.
2. No separate fault detection is necessary before masking.
3. The conversion from a non-redundant system to a TMR system is straightforward.

**Figure 3:** Triplicated TMR system

\[
R_{\text{sys}} = (R_M R_V) \times 3 (R_M R_V)^2 (1 - R_M R_V)
\]

where, \( R_v \) is the reliability of the voter.
4. DYNAMIC REDUNDANCY

- A system with dynamic redundancy consists of several modules but with only one operating at a time.
- If a fault is detected in the operating module it is switched out and replaced by a spare.
- It requires consecutive actions of fault detection and fault recovery.

![Diagram of dynamic redundancy scheme](image)

**Figure 4:** Dynamic redundancy scheme

- A dynamic redundant system with $S$ spares has a reliability:

$$R = 1 - (1 - R_m)^{(S+1)}$$

where $R_m$ is the reliability of each module, active or spare in the system. This reliability function is obtained assuming that the fault detection and the switchover mechanism are perfect.

- The reliability $R$ is an increasing function of the number of spare modules.

![Graph showing reliability of a simple system](image)

**Figure 5:** Dynamic system reliability as a function of simplex system reliability

- However, the use of too many spares may have a detrimental effect on the system reliability.
- Losq has shown that for every dynamic redundant system there exists a finite best number of spares for a given mission time:
  - When the mission time is extremely short is equal to one spare is best.
  - When the mission time is less than one-tenth of the simplex (i.e. non-redundant) mean-life is equal to five spares or fewer is the best.
- The detection of a fault in the individual modules of a dynamic system can be achieved by using one of the following techniques:

1. Periodic tests:
   - Offline.
   - Disadvantage: cannot detect temporary faults unless they occur while the module is tested.

2. Self-checking circuits:
   Provide a very cost effective method of fault detection
3. **Watchdog timers:**
   Timer, checkpoints

- **Reconfiguration:** switching the faulty element and selecting the system output to come from one of the alternative modules.

- **Retry:** so that a module will not be removed because of a temporary fault.

- **Self-repair:** the replacement is invisible to the user and the system continues its operation uninterrupted.

- In general dynamic redundant systems can be divided into two categories:
  (a) Cold-standby system.
  (b) Hot-standby system.

5. **EXPERIMENTAL RESULTS**
6. CONCLUSION

Fault-tolerance is achieved by applying a set of analysis and design techniques to create systems with dramatically improved dependability[3].

As new technologies are developed and new applications arise, new fault-tolerance approaches are also needed. In the early days of fault-tolerant computing, it was possible to craft specific hardware and software solutions from the ground up, but now chips contain complex, highly-integrated functions, and hardware and software must be crafted to meet a variety of standards to be economically viable.

Fault-tolerant computing already plays a major role in process control, transportation, electronic commerce, space, communications and many other areas that impact our lives. Many of its next advances will occur when applied to new state-of-the-art systems such as massively parallel scalable computing, promising new unconventional architectures such as processor-in-memory or reconfigurable computing, mobile computing, and the other exciting new things that lie around the corner.

References


