Chapter 22
Building Dependable Electronic Systems for Autonomous Maintenance

Richard McWilliam, Philipp Schiefer and Alan Purvis

Abstract Maintenance repair and overhaul (MRO) of high value systems is expensive, time consuming and relies heavily upon back-to-base repair and overhaul activity. Autonomous maintenance of repairable systems is a rapidly developing area in through-life engineering services that specifically aims to reduce both mean time to repair and frequency of preventative maintenance. Modern engineering systems must perform reliably in the event of random upset events that threaten to induce malfunction or unpredictable behavior. These requirements are fuelling the integration of fault-tolerant and self-repairing techniques into electronic systems at design time. This chapter investigates emerging techniques being utilised in electronics that bring new self-repair capability to high-value applications such as aviation, land vehicles, renewable energy and space exploration. The cost/benefit trade-off of self-repair strategies is analysed in terms of redundant resource allocation and key performance metrics. The potential for future uptake is discussed in the context of current and next-generation platforms.

22.1 Introduction

The emergence of built-in self-test (BIST), self-reconfiguration and mechatronic assist technologies is driving new research and development of systems that will be capable of self-repair. Within the electronics domain these concepts promise to create the ability to maintain in-service operation in the presence of faults by either masking the effects of the fault or else performing self-reconfiguration to remove the faulty logic. The former is useful for handling random, non-permanent fault events while the latter re-establishes a fault-free system by deactivating logic that has suffered permanent faults. The primary benefit is that such systems become

R. McWilliam (✉) · P. Schiefer · A. Purvis
Centre for Electronic Systems, School of Engineering and Computing Sciences, Science Laboratories, Durham University, South Road, Durham DH1 3LE, UK
e-mail: r.p.mcwilliam@durham.ac.uk

L. Redding and R. Roy (eds.), *Through-life Engineering Services*,
Decision Engineering, DOI 10.1007/978-3-319-12111-6_22

better able to look after themselves by performing self-maintenance tasks, thus leading to increased availability.

A common feature present in all self-repair strategies this is that of additional redundant resources, implemented either at the fine-grained level (e.g., transistor or gate element) or modular sub-component level (e.g., chip or board level). Redundancy has been deployed to great effect in increasing the yield of electronics manufacturing processes for many years, especially for high density components such as memory chips. In recent years however there has been significant rejuvenation of established fault masking techniques for increased fault tolerance following the emergence of next generation nano-electronic fabrication techniques that will suffer lower yield than current semiconductor processes. Aside from manufacturing yield, there is significant interest in supporting through-life maintenance through the provision of self-repairing capability by incorporating self-repair by autonomous reconfiguration within both COTS FPGAs and custom ASICs.

The basic concepts of fault masking and self-reconfiguration suitable for electronics design are discussed in Sect. 22.2. Section 22.3 investigates the different ways in which redundant resources may be deployed and identifies key performance metrics. State of the art strategies that utilize COTS FPGA chips are discussed in Sect. 22.4. Finally, a brief commentary on the future of self-repair and key challenges is provided in Sect. 22.5 along with a qualitative summary of key performance metrics.

22.2 Basic Concepts and Motivation

The incorporation of fault masking, built-in test (BIT) and built-in self-repair (BISR) functionality brings attractive potential benefits: (1) reduction of the cost associated with MRO by assisting with efficient maintenance scheduling; (2) increased availability of electronic components and systems; and (3) extension of the predictable operational life time by better regulation of wear out. The fundamental challenge of implementing test and repair strategies stems from the fact that redundant resources are expensive and must therefore be applied sparingly or else the design quickly becomes unyielding and expensive. Efficient deployment of redundant resources requires design compromise in which the expected cost/benefit trade-off must be understood clearly in order to create an effective mitigation strategy. There is as yet no single optimal strategy for fault-masking or self-repair [1, 2]. Evidence of this is seen in electronic systems that incorporate boundary scan test interfaces, which are in common for production test and repair. This type of interface provides an effective trade-off between resource and benefit for production yield enhancement, however its scope is somewhat limited for in-service duties because the circuitry is not self-contained and relies upon external hardware in order to provide full BIST capability. This special interface is not generally

optimised for low power, but rather high speed due to the fact that in the production environment each second of test time adds significant overhead[1] to product cost [3]. Thus, implementations that enable in-service BIT, fault-masking or BISR tend to rely upon a multitude of strategies that are constrained by minimal power and resource overheads, but which must also provide effective fault recovery and reporting. In the case of non-repairable faults being detected in the field, human interaction is often necessary in order to confirm faulty hardware to repair or replace the offending component or sub-component and such procedures are routinely carried in the repair shops. BIT, BISR and fault masking offers the potential for in situ diagnosis, fault monitoring and repair.

A fundamental difference between fault masking and self-repair is that the latter requires active management of redundant resources. This is related to practice of dynamic redundancy, in which spare components are available but held in a standby state. When a fault condition is detected a fault detection and reconfiguration unit initiates replacement of the faulty component. This type of strategy can be combined with other fault masking methods, for example triple active redundant hydraulic power systems for aircraft that also incorporate an emergency standby system.

BISR design relies upon an improved understanding of the characteristics of component degradation and random upset events and a suitable strategy for deployment of redundant resources. This challenge has been identified in the context of avionics [4] but is seen in a wide variety of high value systems. A detailed analysis of the trends in reliability of high density SRAM was carried out by White [5] in order to better understand actual failure characteristics for SRAM chips. The result was an apparent departure from the conventional "bath tub" curve (Fig. 22.1), influenced in part by improved understanding of counter-wear out

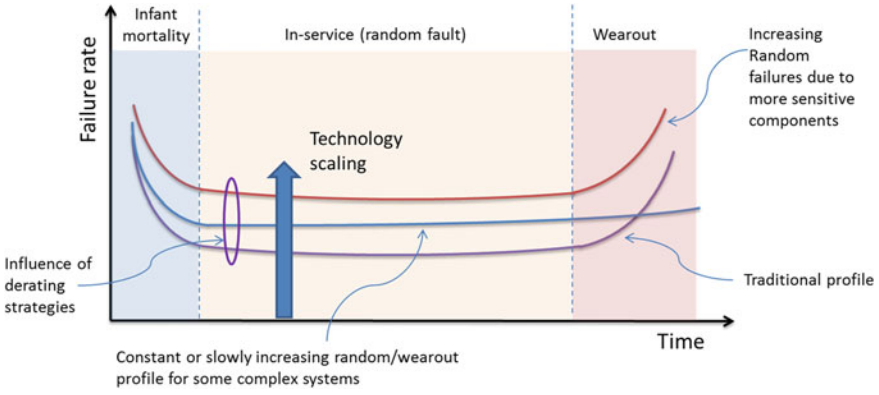

Fig. 22.1 Depiction of potential failure rate versus time profiles expected for modern electronic systems

[1] An often-quoted rule of thumb is that each hour of test time contributes 5 cent to product cost.

measures such as derating, and partly by careful modeling of fault models for production faults, wear out faults and random upset events. In some cases it becomes difficult to distinguish between particular wear out and random failures that assume nominally constant rate during in-service operation. This is a very important feature in the deployment of self-repair strategies because the selection of optimal mitigation strategy requires intimate knowledge of the underlying fault characteristic.

It is also important to keep in mind the robustness of wholly integrated systems that require sensors and actuators in addition to core circuitry. A holistic view should be formed whereby fault tolerance of sensor data and output data is included into the redundancy scheme whenever possible. A traditional approach here is to incorporate concurrent error detection using information redundancy, such as error correction codes (EDC) that seek to filter incoming sensor data and remove faulty code streams.

22.2.1 Sources of Errors

Electronic systems are bombarded by a variety of error-inducing events. Random errors may be caused by environmental factors such as hostile electromagnetic interference (EMI), high energy radiation particles and thermal cycling. Mitigation against EMI is provisioned at design time by applying appropriate grounding, shielding and transient suppression. Random events such as electromagnetic pulses (EMP) cause errors that are difficult to predict. High energy particles interact with semiconductor junctions and are also fundamentally random by nature [6]. Their influence can become of great concern for space and high altitude applications, a classic example being spurious pixels appearing in satellite imaging sensors [7]. There are however growing concerns for the susceptibility of ground-based systems, particularly those relying on SRAM chips since the error rate increases inversely with transistor scaling [8].

22.2.2 Deployment

A natural progression of BIST is of course to begin furnishing the circuitry with resources that enable BISR capability. This may take the form of either fault masking or self-reconfiguration techniques of which there is a great variety. In order to understand their relative merits, key performance factors need to be understood. There are many examples where redundant resources are added, often in modular fashion, for the purpose of fault masking where the goal is to preserve the correct functionality in the presence of faulty logic [2, 9–11]. While fault masking strategies may not be considered as being truly self-repair, it can be argued that they draw upon redundant resources in order to reassert the correct internal signal and thus

represent a step towards self-repair of *information* rather than *logic*. On the contrary, self-reconfiguration strategies seek to eliminate faulty logic and re-establish a fault-free circuit and are usually more complex due to their need for both test and repair.

22.2.3 Key Performance Metrics

As with any new capability incorporation of self-maintenance requires an evaluation of cost/benefit trade-offs. The key features of self-maintenance have been identified for FPGA and custom ASIC architectures, an example of which is shown in Table 22.1. This classification, adopted from [12], originates from self-repair strategies implemented using FPGAs but the concepts are common to general self-repair strategies.

We now discuss in detail each of the metrics mentioned in Table 22.1.

Physical resources In order to incorporate self-repair capabilities, special resources must be added that may be of different design to that of existing logic design, hence requiring different testing and verification methods. In contrast, implementations for FPGAs commandeer a fraction of the available resources, which are then no longer available for application specific tasks. An example is online fault tolerance in FPGAs that require spare PLBs for built-in test and repair

Table 22.1 Evaluation metrics for self-repair

Metric	Description
Physical resources	Additional resources involved in implementing fault handling capability
Critical components	Additional components that must be assumed fault-free
Fault coverage	The type of fault manifestation to be handled and hardware involved e.g., permanent, transient faults that occur in logic blocks, interconnect, memory, IO resources
Fault capacity	Resources necessary to handle a second additional fault event. Used to quantify efficiency of redundant resources
Detection latency	Time incurred to detect presence and location of fault
Recovery time	Time during which system is not available or else executes at reduced speed while repair is carried out
Performance impact	Reduction of overall system performance as a consequence of including repair-mechanism
Recovery granularity	Smallest constituent part of system that can be repaired
Fault exploitation	Ability to reuse defective resources

that could otherwise have been utilised to increase the performance of the main task.

Critical components In addition to the correctly functioning circuitry, any fault repair or making strategy must reply upon additional fault-free resources that may be called upon in response to a fault. This includes redundant resources and any logic required to carry out repair. Examples include the majority voter in a spatial redundancy scheme and built-in reconfiguration units. Identification of critical components is especially important for reconfigurable chips that are provisioned with a finite set of resources.

Fault coverage Fault coverage refers to the type of hardware protected by the scheme and may include logic, interconnect, IO logic or memory. It may also relate to the essential characteristic of the fault event itself, be it transient or permanent, singular or persistent in nature. The nature of the design dictates the class of fault that can be tolerated, and for each case the most effective strategy is selected. It has been observed that interconnect failures are somewhat neglected in comparison to logic failures [12].

Fault capacity Once furnished with suitable resources the design is able to recover from certain fault events. Fine-grained fault masking strategies are designed to tolerate single random fault events at any location within a logic sub-circuit, but do not address multiple persistent fault events that accumulate over time. Thus their fault coverage is usually limited to random SEU or single permanent fault events. By contrast, self-repair strategies target cumulative errors by continuously circumventing faulty logic at the expense of consuming resources. As such their fault capacity is often quantified according to the necessary elemental redundant resource needed to address an additional fault. This resource may be allocated at design time (spare configurations) or else enlisted at runtime (spare configurable logic blocks).

Detection latency Many repair strategies need dependable fault detection circuitry in order to generate efficient repair configurations. Detection latency is an important metric defined as the period of time elapsed between the occurrence of a fault (during which the system is potentially untrustworthy) and completion of the fault repair operation. Fault detection must become an intrinsic design element and be optimised for the particular repair strategy. Also it is not necessarily the case that lengthy detection time will result in poor overall repair performance: roving self-testing areas (STARs) [13] incur high detection latency however faults are automatically quarantined once detected, enabling normal execution to continue whilst repairs are carried out.

Recovery time The speed of repair is critical in most applications, especially if when the system clock is frozen during repair. This results in a period after the fault event during which the outputs are inoperative or untrustworthy. An attempt is made by Parris to compare the recovery time for different FPGA-based strategies [12]. However, establishing a clear distinction between detection latency and recovery time can be challenging in some cases.

Performance impact Fault masking often achieves little or no performance reduction in the event of faults but at the expense of significant redundancy overhead. Self-repair strategies attempt to use resources in a more strategic manner,

although performance degradation is still common. Normal operation may be halted whilst the repair is carried out or by conversion of existing resources into redundant resources.

Recovery granularity The smallest constituent part that is repairable using redundant resources is specified by the recovery granularity. For example, TMR is often applied at the component level where each module comprises a complex electronic sub-system such as a flight control computer. In this case repairs proceed by masking or replacing a faulty module and further investigative work is then needed to identify the precise cause of fault. Fine-grained approaches operate at the gate or even transistor level and offer a more diverse variety of repair mechanisms.

Fault exploitation Operates at the logic block, gate or transistor level, where it is possible to detect that a stuck-at or bridge fault has occurred. In this case it is sometimes possible to reuse the offending fault signal within the existing design, e.g. when a stuck-at high level does not affect the output behavior. Self-repair strategies for FGPA platforms are able to achieve fault exploitation, particularly when based upon evolutionary algorithms.

22.3 Deployment of Redundant Resources

Modern maintenance procedures rely upon BIT mechanisms coupled with human maintenance actions in order to find faults and carry out repair operations. In the production environment test and repair is often automated since the conditions are carefully controlled. An example of this is the test and repair of electronic memory chips containing defective transistors that would otherwise be discarded. The procedure involves detection and bypassing of defective cells using spare redundant cells. In order to achieve test and repair capability special circuitry specifically designed for BIST is inserted into the chip design that enables execution of test patterns within the chip and results to be relayed to external test equipment. The results provide an indication of defective cells that are subsequently deactivated and replaced by spare cells, often by laser-activation of fuses for a permanent reconfiguration. This process has many variations and boundary scan methods are applied in many production test situations.

Fault masking strategies seek to restore reliable operation in the presence of faulty transistors, gates and cells and, while common in some designs, are not considered by themselves as being self-repairing. They could instead be viewed as initiating restoration of logic signals propagating though the network using redundant elements incorporated at design-time. Redundancy is deployed at many different levels of design. Some examples are depicted in Fig. 22.2, where replication of transistor, gates, design cells (logic units) and modules are applied within a design. The three fundamental types of redundancy are: spatial, temporal and information, each of which may be applied in isolation or in combination.

Aside of fault masking, redundant resources are required by repair strategies that seek to restore a fault-free circuit i.e., to remove the presence of faulty logic. While

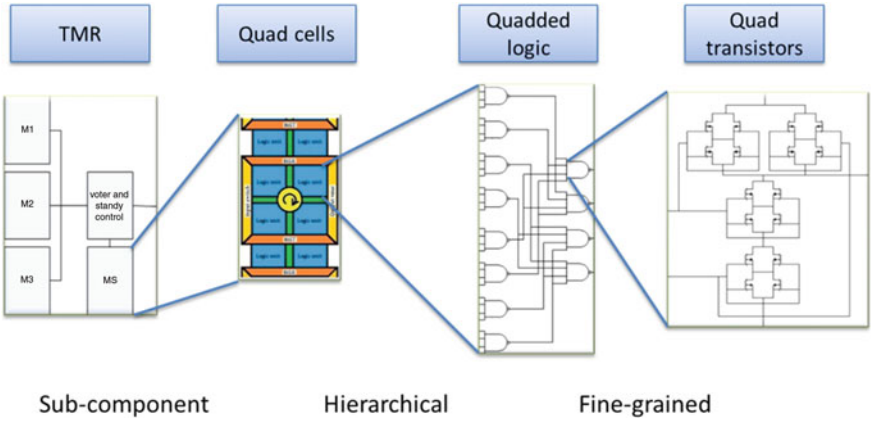

Fig. 22.2 Examples of redundancy deployed at various design levels including transistor, gate, cell and sub-component levels

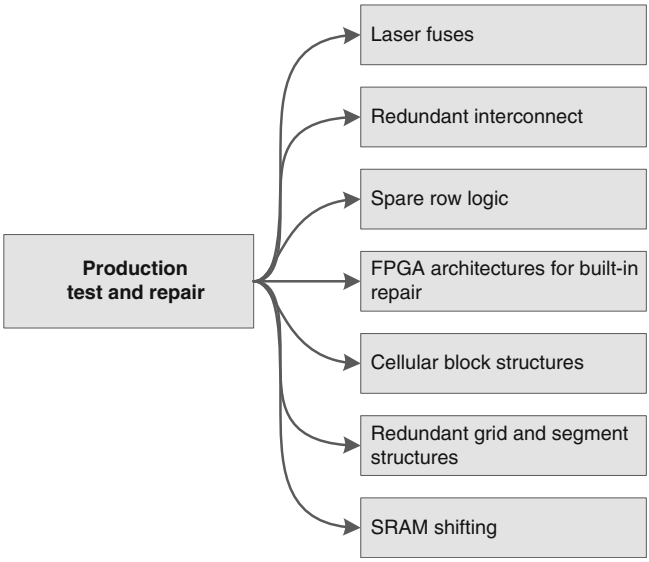

Fig. 22.3 Common repair methods used for yield enhancement (adapted from [1])

fault masking offers the capacity to 'hide' a limited sequence of simultaneous transient or successive permanent fault events, self-repair draws upon additional built-in reconfiguration capabilities to continually restore a fully correct network that is able to sustain repeated faults. A combination of both approaches is clearly desirable in some situations, especially where the operating conditions result in the occurrence of both SEU events and accumulation of permanent faults.

22.3.1 Spatial Redundancy

Spatial redundancy involves direct replication of physical resources that are used to mask faults or to replace faulty logic. Several well-known spatial redundancy schemes exist including triple modular redundancy (TMR), quadded logic and quad transistors. They are particularly attractive for SEU prone conditions since recovery is virtually instantaneous, however their fault capacity is generally low in the presence of cumulative permanent faults. Yield enhancement methods invariably rely upon spatial redundancy in order to provide a fixed set of resources available during production test and repair (Fig. 22.3). Here efforts are directed towards eliminating defects occurring during fabrication rather than random failures or wear out. This leads to a prediction of the defect tolerance of the given circuit that depends on the component failure rate and the effective deployment of redundant resources.

Considering once more random failures that occur in-service, a comparison between the reliability of common spatial strategies can be carried out using simple reliability analysis. A simple example is shown in Fig. 22.4, which compares the redundant strategies of TMR, two spares and quad design. Such comparisons are often analysed in the context of production yield since the probability of component failure, p, can be determined by the process quality. However we may also extrapolate this information into the corresponding in-service domain provided we assume that SEU events are random (i.e., statistically independent) and that the resulting failure condition is analogous to the condition of a manufacturing defect.

In Fig. 22.4 The line labeled "single part" shows the predicted reliability, R, of a non-redundant design. The line labeled "TMR" shows the probability that 2 out of 3

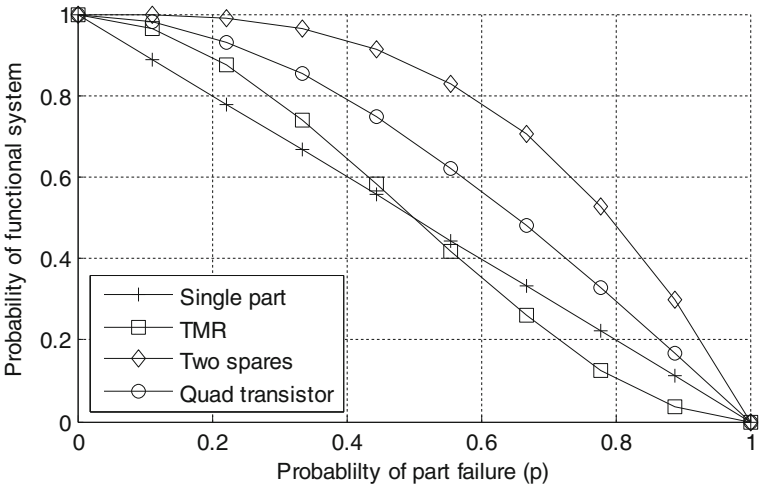

Fig. 22.4 Reliability characteristics for common fault tolerant design strategies, depending on probability of component failure, p

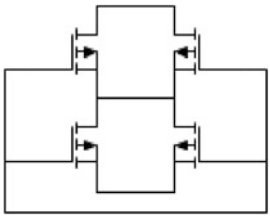

Fig. 22.5 Quad transistor circuit topology

parts are functional and does not take into account the reliability of the majority vote logic. In this case:

$$R = (1-p)^2 + 3p(1-p)^2. \tag{22.1}$$

A further method labeled "Two spares" is also shown, which requires at least 1 out of 3 functional components:

$$R = (1-p)^2 + 3p(1-p)^2 + 3p^2(1-p)^2. \tag{22.2}$$

Equations 22.1 and 22.2 are evaluated by the well-known Binomial distribution [14] applied to system level reliability analysis. Finally, a method specific to transistor level fault tolerance labeled "Quad transistor" is calculated (see also Fig. 22.5) that that requires 3 out of 4 transistors to be functional. In this final case, the reliability is derived as:

$$R = 1 - \frac{3}{2}p^2 + \frac{1}{2}p^3. \tag{22.3}$$

By taking into account the individual suck-at high (SaH), stuck-at low (SaL) and bridging failures, El-Maleh et al. [10]. further examined the theoretical reliability of similar strategies that use N^2 redundant transistors.

Comparisons such as those in Fig. 22.4 lead to several observations. The TMR approach is most effective for small values of p and a poor choice for $p > 0.5$. The two spares approach does yield higher reliability, however this approach does not provide error detection and should be considered inferior to TMR for in-service fault handling (assuming that a reliable majority voter is available). Spares are commonly used in the production environment wherein a sophisticated test machine is able to detect faulty components and replace them with available spares. The quad transistor method exploits the failure characteristics of a quad network such that stuck-at conditions do not cause overall failure. Explicit fault detection does not occur however. This comparison demonstrates the importance of understanding the properties of the applied mitigation strategy and application requirements.

Fault masking will become especially relevant for next generation nanoscale technologies, where the basic resources of transistors and interconnections will be

fabricated with high density but subject to lower low yield than achieved by current manufacturing processes. This has led to considerable activity in the area of online fault tolerant methods, which use massive redundancy to bring improvements to overall system availability [14, 15]. These strategies could be viewed as achieving the creation of reliable circuits in the presence of many faults as per Von Neumann's early work on the principle of building reliable computational networks. The primary benefits are twofold: built-in fault tolerance at the point of manufacture, and SEU fault masking. Note however that the allocation of redundant resource is allocated at design-time and hence there is a fixed resource for both manufacturing test and repair and in-service fault tolerance.

Fine-grained approaches In the case of fine-grained strategies such as the N^2 transistor design circuit, fault rate modelling should take into account behavior at both underlying transistor structure and gate level. In [10] it was demonstrated that fine-grained redundant methods offer favorable failure rates in comparison to gate and cell based redundancy for certain designs. An alternative detailed gate-level fault model presented in [16] accounts for transistor and interconnect level faults specific to CMOS NOR gates. This was used to build an accurate fault injection model for CMOS fault rate analysis. The properties of a gate-level redundancy scheme were investigated in [17] by adopting a simple fault injection model and insertion of various open and short circuit conditions. This enabled evaluation of detectable and undetectable fault conditions and potential repair mechanisms.

22.3.2 Temporal Redundancy

Circuits that incorporate temporal redundancy generate majority signals with minimal hardware by repeated use of logic units to calculate several results over time. If no faults have occurred during the time frame, then a fault-free network is assumed. This leads to reuse of physical hardware but requires a minimum time period before a valid majority vote signal becomes available. Hence detection latency can be considerable. Temporal and spatial redundancy can be combined in order to provide more flexible self-checking capabilities. For example, repeated operations across multiple identical hardware cells provides additional integrity checking beyond purely TMR implementations.

22.3.3 Information Redundancy

Concurrent error detection relies upon information redundancy in the form of additional information added to data patterns stored in memory. This enables recovery from corrupted data bit exploiting additional information redundancy placed into the data pattern. The additional information is referred to error detection

and correction (EDC) codes. Commonly implemented for communication channels, this approach has also been implemented to protect embedded hardware design. In distributed cellular architectures, where special codes analogous to DNA sequences are stored locally in every cell, reconstruction of the correct data occurs continuously even in the presence of multiple upsets [18].

For digital logic whose functional behavior captured in the form of a state transition table, EDC can be applied to protect the data (and hence the functional behaviour). Finite state machines (FSMs) may be protected in this way mapping their state table to a suitable look-up table (LUT) stored in regular memory, which can in turn be protected using EDC codes [19]. When stored in read-only memory (ROM), high-speed compact hardware implementations result although additional (and vulnerable) execution logic is needed to implement the memory address look up and input/output interfacing. ROM-based FSMs have become prevalent in ASIC design [20] but also in FPGA implementations due to their speed and compactness [21]. Even so, the additional logic necessary to run the state machine is not negligible and must also be protected from faults. An elegant solution here is state mapping, where the original state codes themselves are modified to include single error correction (SEC) redundancy codes. This implementation, illustrated in Fig. 22.6, is able to tolerate SEUs present in the LUT and in the next state logic but cannot directly protect the EDC logic used to remove single errors present in the next state. Rochet observed [22] that the vast majority of random errors occurring in the EDC logic are fixed by the single error correction (SEC codes), however the resource overhead associated with additional LUT entries and the error decode logic leads to ever more complex designs [19]. This can ultimately lead to poor overall resource deployment in contrast to simpler strategies such as TMR, therefore careful consideration of the repair method is once again important.

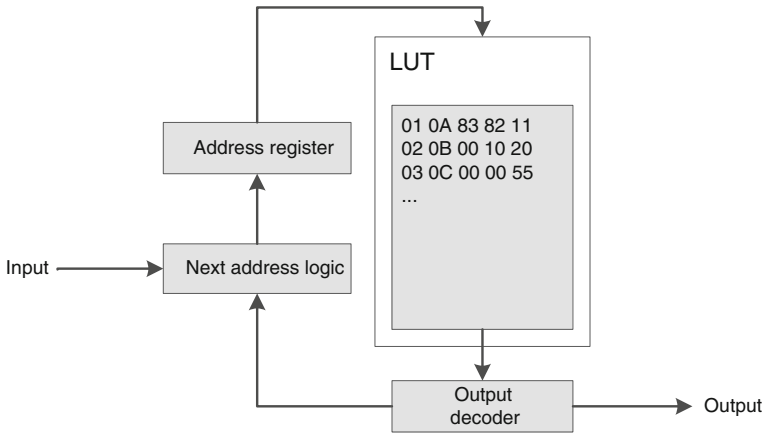

Fig. 22.6 Block diagram of ROM based FSM with LUT contents protected by EDC codes. The FSM requires minimal execution logic for next address generation and input/output interfacing

22.4 Platforms for Self-repair

Before examining the features specific to self-repair strategies, is it useful to briefly consider recent developments in compatible hardware platforms. The most common platform is the field programmable gate array (FPGA), which is essentially an ASIC chip furnished with a large pool of sophisticated reconfigurable logic resources, memory and interconnect routing. These chips are popular due to their in situ reconfigurability, wherein a bitstream file is loaded into chip in order to organise its resources at runtime. The resulting die is extremely densely populated with logic, SRAM and routing resources necessitating extensive production test and repair in order to enhance manufacturing yield. Because of this device-level (DL) fault tolerant methods are used that are transparent to the end user. To overcome this imitation, a large variety of configuration-level (CL) methods have been devised that seek to reuse the FPGA's resources for dynamic in-service repair [1, 2, 12]. Reconfigurations are carried by alteration of the configuration bitstream. Alternative full custom chip designs have been also proposed that are furnished with new redundant resources specifically tailored for fault mitigation rather than production test and repair. Strategies include fine-grained [23] and cell-based [24] redundancy. Another class of ASIC is the fully customised chip having a non-reconfigurable design that is constrained according to speed, efficiency and size restrictions. These designs may be equipped with fault tolerant resources at design-time however their target application is more likely to be performance sensitive hence highly optimised implementations are needed.

22.4.1 Key Strategies

Self-repair requires a combination of BIT and self-reconfiguration in order to detect and eliminate faulty logic using redundant resources. The principle of self-repair as autonomous design has been discussed at length [25], and principally involves the actions of detect-divert or detect-replace in order to circumvent fault logic. Another potential feature is self-preservation, which attempts to inhibit future degradation. Efficient maintenance scheduling relies on having a detailed knowledge of expected fault events and actions to be taken. An important feature of self-repair strategies is the provision of built-in logging (and possibly classification) of both random fault events. Accurate records of SEU and permanent errors are thought to be extremely valuable in gauging the remaining fault capacity and for refinement of overhaul work scheduling.

Considering strategies that are specific to FPGAs, numerous comparisons of resource-performance trade-offs have been reported, in most cases by estimating the key metrics described in Sect. 22.2.3 (primarily relating to detection latency, recovery time and resource overhead) [1]. A comparison of the relative benefits has also been presented in the form of a number of "sustainability metrics" that include

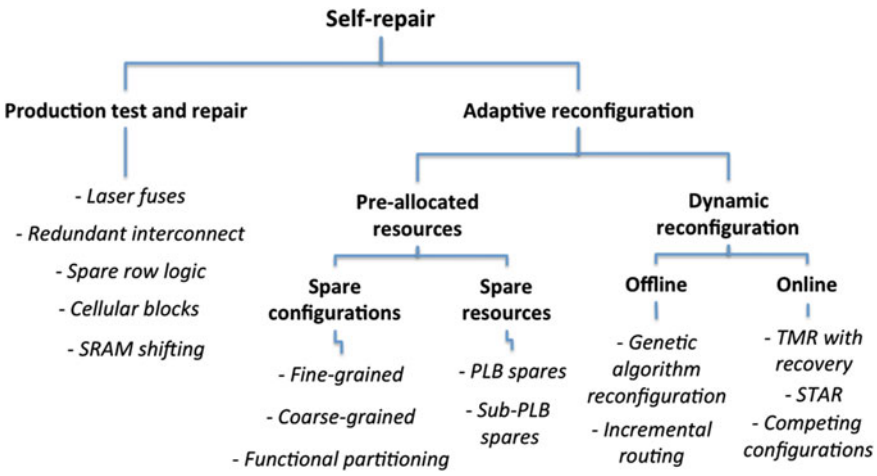

Fig. 22.7 Strategies for self-repair at point of manufacture and for in-service reconfiguration (see [1, 12, 27]). *PLB* Programmable logic block; *SRAM* Static random access memory; *STAR* Self-test area

fault capacity, granularity, interconnect fault handling and critical fault resources [12]. A further survey has considered fault detection as being an essential part of the self-repair process [26]. Findings from these studies have suggested that no single method is optimal and that the application priorities must be carefully evaluated in order to select an effective fault tolerant approach. This is true not only FPGAs but even more so for custom ASIC designs that must be optimised according to power and speed constraints. Figure 22.7 provides a summary of prominent methods reported in the literature, classified according to strategy. Pre-allocated and dynamic reconfiguration methods feature heavily in FPGA strategies, where their ability to alter the configuration bitstream is exploited to circumvent faulty logic. Pre-allocation utilises alternative configurations defined at design time in the form of spare configurations or spare logic. Spare configurations are allocated at design time and comprise a collection of alternative bitstream configurations that can be loaded in response to a faulty condition. These schemes depend upon effective BIT in order to determine the most appropriate replacement configuration that is most likely to circumvent the faulty logic. Similarly, spare resources are made available at design time and are activated upon detection of a faulty logic block. This results in a direct substitution of the block with a nearby spare block. Spare blocks are arranged at a finer granularity than spare configurations but their distribution must be allocated carefully in order to provide sufficient fault capacity.

By contrast, dynamic methods are able to generate new configurations in the field as a direct response to faults. Offline methods seek to derive a new configuration using an algorithm that attempts to process fault information, assess available resources and generates a new configuration. An FPGA's normal operation must be halted in order to reconfigure the device and in some cases to provide additional

on-chip resources for calculation of the new configuration. Genetic algorithms have also been demonstrated to be capable of achieving impressive fault capacity with efficient resource usage and even fault exploitation. However, their recovery time tends to become unbounded due to the nature of the algorithms involved. Online methods seek to preserve active operation while BIT and reconfiguration operation are carried out. These approaches are somewhat complex due to their need for real-time bitstream manipulation. That said, the TMR with recovery method simplifies the complexity at the expense of performance reduction due to the need for high spatial redundancy within the active configuration. Application performance is also compromised by the STAR method approach. A detailed examination of this can be found in [13], where the performance penalty for a various design using STARs is quantified in terms of maximum clock speed and spare resource overhead. For the sample designs considered, the maximum design clock speed was reduced by some 2.5–15.1 % when 20 % of the chip was reserved for STARs. However, this penalty is countered by important benefits including the ability to deal with of dormant faults by adaptive re-usage of resources and incremental fault tolerance that escalates the provision of redundant resources as faults become more aggressive and frequent.

22.5 Potential Impact and Uptake of Autonomous-Maintenance

To date, no single fault masking or self-repair strategy has proved all encompassing; each particular design option must be evaluated in order to determine the best strategy. A wider understanding of cost/benefit trade-offs at different levels design level granularity of would be of great benefit in this area. A key issue is that accurate evaluations of fault rate behavior generally lead to complex and protracted analysis, resulting in lengthy design verification. This may be alleviated by developing useful optimisation metrics capable of exploring complex combinations of redundant strategies and application constraints. This process should begin by evaluating key metrics against the top-level strategies of fault masking and self-repair, then classifying the benefits brought about through each specific approach.

Table 22.2 provides a starting point for this process, however a common resource allocation model would be needed in order to generate different combinations of redundancy strategies.

Whether the characteristics of a particular self-repair strategy are considered acceptable depends on the application and the cost/trade-offs evident in Table 22.1. By example, the STAR strategy exerts a long detection latency amounting to several seconds typical for complex roving BIT mechanisms. However since test and repair coverage is 100 % and utilises only non-active logic at any single point in time, there is no actual down-time incurred for the self-checking process as far as the active logic is concerned. Should a fault strike then the worst-case effective

Table 22.2 Comparison between key metrics of fault masking and self-repair strategies

Metric	Fault masking	Self-repair
Mitigation strategy	Faulty logic is tolerated without error in output behavior	Faulty logic is circumvented or reused
Fault capacity	Limited capacity for SEU errors. Resources may be able to handle second additional fault—but only in limited locations	Resources designed to generate significant fault capacity
Detection latency	No explicit detection provided, but majority vote errors could be used	Varies from 100's μs to many seconds
Recovery time	Virtually instantaneous	Between 10's of milliseconds and several seconds (or even unbounded)
Fault reporting	Not inherent, but majority vote offers limited reporting	Event data available, but most solutions not sufficiently developed for reporting or logging

detection and recovery time could be unacceptable long, and redundant fault-masking resources may be needed at a more finely-grained design level. Conversely TMR fault-masking strategy offers negligible detection and recovery time however requires expensive triplicate structures and reliable majority voting logic in comparison to around 10–20 % spare resource for STAR implementations.

Another factor is the necessary complexity of the underlying fault-tolerant resources: FPGAs commonly employed for self-repair strategies are composed of blocks containing complex arrangements of logic and interconnect (albeit with a well-understood and regular structure). Fault masking strategies tend to be implemented using very simple logic that is no more complex than the non-redundant design. N-modular redundancy principles have been reported at the extremely fine-grained transistor and gate level [17]. The TMR principle is sufficiently flexible for FPGA design that use multiple configurations when combined with standby configurations [28]. Indeed, the replication of entire components or sub-components is commonly adopted in mission critical situations such as navigation and mission control computers, although there is great interest in exploring less resource-expensive strategies that achieve equivalent reliability through self-repair using COTS components.

22.5.1 Test and Verification

There are three key aspects testing of self-repair methods. First, verification of the core functionality is necessary in order to ensure that the fault tolerant mechanism operates as intended. Second, an evaluation of key metrics (Sect. 22.2.3) will lead to better understanding of resource/performance trade-offs. Third, test and verification of the complete sub-component is needed in order to ensure that its behavior is predictable. Self-repair systems must be able to adapt and continue in the presence of faults, as well as maintain predictable behavior at all times. For

mission-critical applications autonomy raises concerns regarding predictability in addition to serviceability and certification, requiring that new standards be developed that set out suitable qualification procedures. The concept of design for *full fault coverage* using built in test is relevant here [29]. In production test, BIT logic must be able to cover every critical test combination in order to provide full repair of defect logic. Similarly, complete coverage is necessary when testing fault masking or reconfiguration strategies.

A useful technique for testing performance and compliance of autonomous self-repair is fault injection. This takes the form of asserting fault conditions within the circuit in a random fashion in order build up a statistical picture of its fault behavior. For example, the fault injection algorithm proposed in [30] involves temporarily setting fault conditions in a sequential manner where it is assumed that the DUT design remains unchanged during the test procedure. Chip production and in-field diagnostics exploit boundary scan logic [31] which can be used for testing of self-repair mechanisms. However the test at system level can become complex since appropriate resources must allocated for built-in test and control lines [32]. Production yield test and repair machines utilise highly optimised test-repair algorithms, allowing them keep track of permanently faulty logic resulting from manufacturing defects. Although the test requirements appear similar in nature, online reconfiguration strategies seek to alter the configuration of the DUT in response to multiple fault events, some of which may be permanent by nature. Hence a suitable fault injection tool must create a combination of transient and permanent fault conditions, and keep track of the complex sequence of faults during test. Due to the complexity of design for self-repair, proxy hardware is often employed during development that simulates the design before committing to ASIC manufacture. This is generally achieved using FPGA platforms that are configured to emulate the design. Whether used for emulation or test bench interfacing, the underlying proxy hardware must be sufficiently reliable and bug-free so that it may be assumed error-free.

22.5.2 Potential Impact and Uptake

Within the context of assisted-maintenance the problem addressed by self-repair strategies is that of increasing the robustness of sub-components that would otherwise perform unpredictably or else fail entirely in the presence of random and permanent failures. In the latter case, the hope is to achieve a graceful degradation that prolongs operational lifetime and provides valuable monitoring of remaining fault capacity as an indication of impending end of life at the sub-component level. Strategies for self-repair draw partly upon techniques developed for production yield enhancement. This means that the end customer is never aware of production-related repairs. However self-repair strategies must operate without access to external test hardware equipment and thus maintaining customer transparency is more challenging. In some instances the repair strategy relies on customer-level

tools such as vendor-supplied synthesis and layout for FPGAs. It could be argued however that a sustainable model (in the context of predictability, serviceability and certification) is one in which fault tolerance operates at a guaranteed "self-repair hardware level" that is transparent to the customer and hence may be regarded as autonomic. Such concepts are especially conducive to custom ASIC designs.

Ongoing developments in autonomous maintenance aim to achieve in-service "on the wing" active repair during operation or perhaps during predicable periods of downtime. They would no longer be limited to scheduled maintenance periods. Aside from those online reconfiguration methods discussed (Sect. 22.4.1), there are other examples of in-service models. For example a self-tuning analogue RF circuit has been demonstrated capable of online self-correction actions and thus is able to sustaining optimal performance without minimal degradation [33]. In this case an in situ BIT mechanism is incorporated that constantly monitors and performs self-correction of parameters that would otherwise drift out of specification. This example might be considered as being self-optimising rather than self-repairing, however the concept is similar since both rely on self-contained detect-restore actions.

Briefly revisiting the classic model of component lifetime discussed in Sect. 22.2, for some systems random and wear out failures are difficult to differentiate and hence great care is needed when allocating self-repair strategies such that they target the correct failure mechanism (i.e., presumably prioritising random rather than wear out failures). With this mind, we must ask the question: how to we select the most appropriate mitigation strategy and prove its efficacy? Clear and conclusive assessment of the metrics discussed above combined with effective test and verification are essential.

The next question is then: who should carry out this task? This is a complex systems design problem since it is not obvious as to what level of autonomy and transparency is most beneficial. Abstraction from application specific development tools may seem appropriate in terms of resource allocation, however acceptable limits on impact to performance, power and timing are very much application dependent. One final design aspect is that self-maintaining systems will be expected record and report fault history and thus assist planning of MRO scheduling. Fault event logging is already present in many systems, for example in automotive Engine control units (ECUs). However, additional information is needed such as remaining fault capacity and should be reported in a sensible format that is easily understood. An example would be when a sub-component that is experiencing aggressive fault conditions, necessitating an escalation of resource allocation in order to maintain fault-free operation. Reporting of remaining capacity (and indeed trending of this information) provides valuable information to both maintenance scheduling and decision-making about when to replace depleted resources. An key benefit is be the ability distinguish between exhaustion of fault capacity and general wear out, which would better inform decisions involving whether to replace complete sub-component or to whether to carry out overhaul to replace depleted boards and modules contained therein.

Our demand for ever-increasing performance and efficiency of engineering system is driving the adoption of COTS components even for mission critical applications. This is fuelling the uptake of concepts in self-repair and autonomy that for example achieves similar design robustness to that achieved through component screening. Self-repair concepts are also generating significant interest in the area of next-generation nano electronics where transistor and interconnect reliability is expected to be much lower than of current technology. In the widest context, self-repair seeks to increase the robustness of cost-sensitive, high value systems and therefore bring about cheaper maintenance through autonomous strategies. However a better understanding the cost/benefit trade-offs and effective design methodology is essential for its uptake.

References

1. Cheatham JA, Emmert JM, Baumgart S (2006) A survey of fault tolerant methodologies for FPGAs. ACM Trans Des Autom Electron Syst. 11(2):501–533. doi:10.1145/1142155.1142167
2. Morgan KS, McMurtrey DL, Pratt BH, Wirthlin MJ (2007) A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs. IEEE Trans Nucl Sci 54 (6):2065–2072. doi:10.1109/TNS.2007.910871
3. Evans AC (1999) Applications of semiconductor test economics, and multisite testing to lower cost of test. In: IEEE proceedings of international test conference. doi:10.1109/TEST.1999.805620
4. Walter JD (2003) Methods to account for accelerated semi-conductor device wear out in longlife aerospace applications. DTIC document, Virginia
5. White M (2010) Scaled cmos technology reliability users guide. http://hdl.handle.net/2014/414912010
6. Sexton FW (2003) Destructive single-event effects in semiconductor devices and ICs. IEEE Trans Nucl Sci 50(3):603–621. doi:10.1109/TNS.2003.813137
7. (2007) Radiation Effects on Embedded Systems, 1 edn. Springer, Berlin
8. Nicolaidis M (2010) Soft errors in modern electronic systems. Springer, Berlin
9. Von Neumann J (1956) Probabilistic logics and the synthesis of reliable organisms from unreliable components. Automata studies 34:43–98
10. El-Maleh AH, Al-Hashimi BM, Melouki A, Khan F (2009) Defect-tolerant n2-transistor structure for reliable nanoelectronic designs. IET Comput Digital Tech 3(6):570–580. doi:10.1049/iet-cdt.2008.0133
11. Lyons RE, Vanderkulk W (1962) The use of triple-modular redundancy to improve computer reliability. IBM J Res Dev 6(2):200–209. doi:10.1147/rd.62.0200
12. Parris MG, Sharma CA, Demara RF (2011) Progress in autonomous fault recovery of field programmable gate arrays. ACM Comput Surv 43(4):31. doi:10.1145/1978802.1978810
13. Emmert JM, Stroud CE, Abramovici M (2007) Online fault tolerance for FPGA logic blocks. IEEE Trans Very Large Scale Integr (VLSI) Syst 15(2):216–226. doi:10.1109/TVLSI.2007.891102
14. Breuer MA, Gupta SK, Mak TM (2004) Defect and error tolerance in the presence of massive numbers of defects. IEEE Des Test Comput 21(3):216–227. doi:10.1109/MDT.2004.8
15. Han J, Gao J, Jonker P, Qi Y, Fortes JAB (2005) Toward hardware-redundant, fault-tolerant logic for nanoelectronics. IEEE Des Test Comput 22(4):328–339. doi:10.1109/MDT.2005.97

16. Wadsack RL (1978) Fault modeling and logic simulation of CMOS and M08 integrated circuits. Bell Syst Tech J 57(5):1449–1474
17. Kothe R, Vierhaus HT, Coym T, Vermeiren W, Straube B (2006) Embedded self repair by transistor and gate level reconfiguration. In: Design and diagnostics of electronic circuits and systems, IEEE. doi:10.1109/DDECS.2006.1649613
18. Jones D, McWilliam R, Purvis A (2008) Designing convergent cellular automata. Biosystems 96:80–85
19. Wendling X, Rochet R, Leveugle R (1996) ROM-based synthesis of fault-tolerant controllers. In: IEEE international symposium on defect and fault tolerance in VLSI systems
20. Gerbaux L, Saucier G (1992) Automatic synthesis of large Moore sequencers. Integr VLSI J 13(3):259–281. doi:10.1016/0167-9260(92)90031-S
21. Moreno G, Civit-Balcells A, Guerra-Gutierrez P (2007) ROM-based finite state machine implementation in low cost FPGAs. In: IEEE international symposium on industrial electronics
22. Rochet R, Leveugle R, Saucier G (1993) Analysis and comparison of fault tolerant FSM architecture based on SEC codes. In: IEEE international workshop on defect and fault tolerance in VLSI systems
23. Walker JA, Trefzer MA, Bale SJ, Tyrrell AM (2013) PAnDA: a reconfigurable architecture that adapts to physical substrate variations. IEEE Trans Comput 62(8):1584–1596. doi:10.1109/TC.2013.59
24. Basha C, Pillement S, Lagadec L New reconfigurable fault tolerant FPGA architecture: a design for mission critical applications. 8th HiPEAC workshop on reconfigurable computing, Vienna, Austria, Jan 20–22
25. Frei R, McWilliam R, Derrick B, Purvis A, Tiwari A, Serugendo GDM (2013) Self-healing and self-repairing technologies. Int J Adv Manufact Technol 69:1033–1061. doi:10.1007/s00170-013-5070-2
26. Campregher N, Cheung PYK, Constantinides GA, Vasilko M (2006) Reconfiguration and fine-grained redundancy for fault tolerance in FPGAs. In: International conference on field programmable logic and applications (FPL '06)
27. Stott E, Sedcole P, Cheung P (2008) Fault tolerant methods for reliability in FPGAs. In: International conference on field programmable logic and applications (FPL '08)
28. Zhang K, Bedette G, DeMara RF (2006) Triple modular redundancy with standby (TMRSB) supporting dynamic resource reconfiguration. In: IEEE autotestcon
29. McCluskey EJ, Bozorgui-Nesbat S (1981) Design for autonomous test. IEEE Trans Circuits Syst 28(11):1070–1079. doi:10.1109/TCS.1981.1084930
30. Quinn HM, Black DA, Robinson WH, Buchner SP (2013) Fault simulation and emulation tools to augment radiation-hardness assurance testing. IEEE Trans Nucl Sci 60(3):2119–2142. doi:10.1109/TNS.2013.2259503
31. Sedmark RM (1994) Boundary-scan: beyond production test. In: 12th IEEE VLSI test symposium
32. Arlat J, Aguera M, Amat L, Crouzet Y, Fabre JC, Laprie JC et al (1990) Fault injection for dependability validation: a methodology and some applications. IEEE Trans Software Eng 16 (2):166–182. doi:10.1109/32.44380
33. Goyal A, Swaminathan M, Chatterjee A, Howard DC, Cressler JD (2012) A new self-healing methodology for rf amplifier circuits based on oscillation principles. IEEE Trans VLSI Syst 20 (10):1835–1848. doi:10.1109/Tvlsi.2011.2163953