Formal Probabilistic Analysis of Stuck-at Faults in Reconfigurable Memory Arrays

Osman Hasan, Naeem Abbasi, and Sofiène Tahar

Dept. of Electrical & Computer Engineering, Concordia University
1455 de Maisonneuve W., Montreal, Quebec, H3G 1M8, Canada
{o_hasan,n_ab,tahar}@ece.concordia.ca

Abstract. Reconfigurable memory arrays with spare rows and columns are quite frequently used as reliable data storage components in present age System-on-Chips (SoCs). The spare memory rows and columns can be utilized to automatically replace rows or columns that are found to contain a cell fault after fabrication. One of the biggest SoC design challenges is to estimate, prior to the actual fabrication process, the right number of these spare rows and spare columns for meeting the reliability specifications. Traditionally, computer simulation techniques are used to perform probabilistic analysis of reconfigurable memory arrays but they provide inaccurate results. To ensure accurate analysis and thus more reliable SoC designs, we propose, in this paper, a probabilistic theorem proving approach in the domain of reconfigurable memory array analysis. We present a higher-order-logic stuck-at fault model for reconfigurable memory arrays, based on which, we illustrate the formal verification of some key statistical properties related to the number of stuck-at faults and the repairability condition.

1 Introduction

Embedded memory is the most dominant component in terms of silicon area of any System-on-Chip (SoC) these days. Applications such as mobile communication devices, medical and industrial signal processing and digital switching systems used in computer networks all require large amounts of memory. It is expected that by the end of this decade about 90% of the chip area on a typical SoC will be taken up by memory of one type or the other (Static, Dynamic, Flash, or Content addressable) [15]. These memories on a chip are usually organized in very highly optimized structures in an effort to reduce cost. Extremely small memory cell sizes and the fact that a significant amount of the chip area is taken by compact memory arrays, makes memories more prone to defects during fabrication than regular logic. The defects in a memory can render the whole SoC useless. Even in mature fabrication processes where the defect densities tend to be small, the throwing away of any chip is considered unacceptable because of its adverse effect on yield. Moreover, these defects may also lead to devastating situations when the bug is not caught in the testing phase and the faulty chip is used in a safety critical domain, such as medicine, military and transportation.

M. Leuschel and H. Wehrheim (Eds.): IFM 2009, LNCS 5423, pp. 277–291, 2009.

In order to analyze the effects of memory defects, memory fault models are constructed that describe how a fault in memory might occur and predict the resulting device behavior. There are four main types of faults that may occur in a memory array: stuck-at faults, transition faults, coupling faults, and neighborhood pattern sensitive faults [20]. Stuck-at faults, which occur when a memory cell never changes its state, i.e., it is always stuck in one state, is the most commonly used fault model for analyzing memory arrays and logic. The information gathered from the fault models is utilized for the development of techniques for detecting and repairing memory faults. One such widely used technique is to add some redundancy to the memory array during the design phase. This way even after fabrication, we can repair some of the memory faults by replacing the rows or columns of the memory array containing faulty memory cells with the available spare rows or columns. Memories fabricated with these spare rows and columns are usually termed as *reconfigurable memory arrays*. This technique poses an interesting solution to the memory faults problem but comes with a bigger design challenge of estimating the right number of spare rows and columns for meeting reliability specifications. If a combination of spare rows and columns exists such that all faults from the memory array can be eliminated then such a combination of spare rows and columns is called a *repair solution*, and the array is called *repairable*. The repairability problem of a reconfigurable memory array is similar to the vertex cover problem of the bipartite graph and is known to be an NP complete problem [16]. Thus, probabilistic analysis and some graph theory principles are usually utilized to obtain reasonable solutions [25,18,2].

Today, simulation is the most commonly used computer based probabilistic analysis tool for reconfigurable memory arrays, e.g., see [21,23]. Most simulation based memory array analysis software provide a programming environment for defining functions that approximate random variables for probability distributions. The random elements, such as fault occurrences, in a given memory array are modeled by these functions and the model is analyzed using computer simulation techniques [5], such as the Monte Carlo method [19], where the main idea is to approximately answer a query on a probability distribution by analyzing a large number of samples. Statistical quantities, such as expectation and variance, may then be calculated, based on the data collected during the sampling process, using their mathematical relations in a computer. Due to the inherent nature of simulation coupled with the usage of computer arithmetic, the probabilistic analysis results attained by the simulation approach can never be termed as 100% accurate. Moreover, simulation requires an enormous amount of CPU time for attaining meaningful estimates. We generally need to acquire hundreds of thousands of samples to estimate the desired probabilistic quantities and this fact makes the simulation approach impractical when each sample acquisition step involves extensive computations, which is usually the case for analyzing reconfigurable memory arrays due to their large capacities. Thus, simulation should not be relied upon for the analysis of reconfigurable memory arrays, especially when they are used in safety critical areas, where inaccuracies and inadequacies in the analysis may even result in the loss of human lives.

In the past couple of decades, formal methods [8] have been successfully used for the precise analysis of a verity of hardware and software systems. The rigorous exercise of developing a mathematical model for the given system and analyzing this model using mathematical reasoning usually increases the chances for catching subtle but critical design errors that are often ignored by traditional techniques like simulation. Given the sophistication of the present age memory reconfigurable arrays and their extensive usage in SoCs for safety critical applications, there is a dire need of using formal methods in this domain. However, to the best of our knowledge, due to the random and unpredictable occurrence pattern of memory array faults, the usage of formal methods for their analysis has never been attempted. Some of the major reasons for this include the inability to precisely reason about statistical properties, such as expectation and variance, in the case of state-based approaches and the fear of huge proof efforts involved in modeling and reasoning about random occurrence patterns of memory faults in the case of theorem proving with expressive logics.

We believe that due to the recent developments in the formalization of probability theory concepts [14,11,12,13], we are now at the stage where we can handle the probabilistic analysis of reconfigurable memory arrays in a higher-order-logic theorem prover [6] with reasonable amount of modeling and verification efforts. We illustrate the practical effectiveness of this argument by presenting the higher-order-logic theorem proving based analysis of the repairability problem for stuck-at faults in this paper. Even though, we concentrate on stuck-at faults here, the presented approach is quite general and can be essentially utilized to conduct the analysis of other kinds of memory faults as well.

This paper presents a three step approach for tackling the repairability problem. We proceed by formally expressing a stuck-at fault model for reconfigurable memory arrays in higher-order logic. Our formalization utilizes precise random variable functions to express the random components in the model. Secondly, we utilize our formal model to express and verify statistical properties, such as expectation and variance of the number of faults in terms of memory array and spare rows and columns sizes, as higher-order logic theorems. Finally, this formal statistical information is utilized to formally verify a relation that ascertains that a large square memory array is almost always repairable (with probability 1) if stuck-at faults are independent and identically distributed with a specific probability. This result can now be used to accurately estimate the number of spare rows and columns required for reliable operation against stuck-at faults of any reconfigurable memory array without any CPU time constraints. We have utilized the higher-order-logic theorem prover HOL [7] for this work. The main motivation behind using the HOL theorem prover is the fact that it contains most of the foundational probability theory work that we build upon.

The rest of the paper is organized as follows: Section 2 presents some related work. Section 3 provides an overview of HOL probabilistic analysis related foundations that we build upon to conduct the analysis of reconfigurable memory arrays in this paper. In Section 4, we present our formal probabilistic model of the number of stuck-at faults in memory arrays. This is followed by the formal

verification of some statistical properties and the repairability condition in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

Simulation techniques are very commonly used in the yield and repairability analysis for memory arrays. One such yield analysis tool , described in [23], for integrated circuits containing multiple, possibly different, repairable embedded memories. Pseudo random faults are generated based on memory area, defect density, and fault distribution. Then, using a flexible array model, optimal numbers of spare rows and columns for a given memory are determined. The tool is also used to determine the effectiveness of various repair algorithms. In [21] a Built-in self repair (BISR) technique is presented that merges error correction coding schemes and self repair using spare rows and columns. The technique is validated through simulation and it is shown that for defect densities as high as 10^{-2} % (or when 3% of cells are defective) near 100% memory yield can be achieved and thus is suitable for nanometer CMOS process generations.

When memory sizes become large, analysis through simulation very quickly becomes computationally difficult to handle. Paper-and-pencil based analytical analysis have been traditionally used for such cases. A memory array probability model represents either the occurrence of individual faults or the total number of faults as a random variable and thus allows reasoning about statistical properties. Questions, such as "given a certain fault distribution and number of faults can almost every memory array be repaired", or "with how many faults a memory array can almost never be repaired", can then be answered [2,25,18].

To the best of our knowledge, higher-order-logic theorem proving has never been used for the probabilistic analysis of any memory reconfigurable array so far. Though, some useful research related to the foundations of probabilistic analysis is available in the open literature. Random variables can be formalized and verified, based on their probability distribution properties, using the methodology proposed in [14]. In fact, [14] presents the formalization of some discrete random variables along with their verification, based on the corresponding PMF properties. Building upon Hurd's formalization framework [14], the sampling algorithms of a few continuous random variables have also been formalized and verified [11]. In [12,10], we extended Hurd's formalization framework with a formal definition of expectation. This definition is then utilized to formalize and verify the expectation and variance characteristics associated with discrete random variables that attain values in *natural* numbers only.

Besides theorem proving, another formal method that can be used for conducting precise probabilistic analysis of reconfigurable memory arrays is probabilistic model checking [1,22]. The main idea behind this approach is to construct a precise state-based mathematical model of the given memory array and then utilize this model to exhaustively verify the intended, formally represented, probabilistic properties, such as the probability of number of faults being less than some threshold value in a given memory array. Besides the accuracy of the results,

the most promising feature of probabilistic model checking is the ability to perform the analysis automatically. On the other hand, it is limited to systems that can only be expressed as probabilistic finite state machines or Markov chains. Another major limitation of the probabilistic model checking approach is state space explosion [3], due to which large capacity memories cannot be analyzed using this approach. Similarly, to the best of our knowledge, it has not been possible to precisely reason about statistical quantities, such as variance and tail distribution bounds, using probabilistic model checking so far. The most that has been reported in this domain is the evaluation of a small subset of expected values in a couple of model checkers, such as PRISM [17] and VESTA [24]. Because of the above mentioned limitations, probabilistic model checking is not feasible for analyzing memory array repairability problem as the models are usually large and most of the decision making in this domain is made based on statistical quantities. Whereas, the proposed higher-order-logic theorem proving based approach, allows us to analyze a wider range of memory arrays without any modeling limitations, such as the restrictiveness to Markovian models or the state-space explosion problem, and formally verify statistical properties, as will be seen in the next section.

3 Probabilistic Analysis in HOL

The foremost criteria for implementing a theorem proving based probabilistic analysis framework is to be able to formalize and verify random variables in higher-order logic. Hurd's PhD thesis [14] can be considered a pioneering work in this regard as it presents a methodology for the formalization and verification of probabilistic algorithms in the HOL theorem prover. Random variables can be formalized in higher-order logic as deterministic functions with access to an infinite Boolean sequence $\mathbb{B}^\infty$; a source of infinite random bits [14]. These deterministic functions make random choices based on the result of popping the top most bit in the infinite Boolean sequence and may pop as many random bits as they need for their computation. When the functions terminate, they return the result along with the remaining portion of the infinite Boolean sequence to be used by other programs. Thus, a random variable which takes a parameter of type α and ranges over values of type β can be represented in HOL by the function.

$$\mathcal{F} : \alpha \to B^\infty \to \beta \times B^\infty$$

As an example, consider the Bernoulli($\frac{1}{2}$) random variable that returns 1 or 0 with equal probability $\frac{1}{2}$. It can be formalized in HOL as follows

```
⊢ bit = (λs. if shd s then 1 else 0, stl s)
```

where s is the infinite Boolean sequence and `shd` and `stl` are the sequence equivalents of the list operation *'head'* and *'tail'*. The probabilistic programs can also be expressed in the more general state-transforming monad where the states are the infinite Boolean sequences.

```
⊢ ∀ a s. unit a s = (a,s)
⊢ ∀ f g s. bind f g s = g (fst (f s)) (snd (f s))
```

The `unit` operator is used to lift values to the monad, and the `bind` is the monadic analogue of function application. All monad laws hold for this definition, and the notation allows us to write functions without explicitly mentioning the sequence that is passed around, e.g., function *bit* can be defined as

```
⊢ bit_monad = bind sdest (λb. if b then unit 1 else unit 0)
```

where `sdest` gives the head and tail of a sequence as a pair (*shd* `s`, *stl* `s`).

[14] also presents some formalization of the mathematical measure theory in HOL, which can be used to define a probability function $\mathbb{P}$ from sets of infinite Boolean sequences to *real* numbers between 0 and 1. The domain of $\mathbb{P}$ is the set $\mathcal{E}$ of events of the probability space. Both $\mathbb{P}$ and $\mathcal{E}$ are defined using the Carathéodory's Extension theorem, which ensures that $\mathcal{E}$ is a σ-algebra: closed under complements and countable unions. The formalized $\mathbb{P}$ and $\mathcal{E}$ can be used to formally verify probabilistic properties, e.g.,

$$\vdash \mathbb{P}\ \{\texttt{s | fst (bit s) = 1}\} = \frac{1}{2}$$

where the HOL function `fst` selects the first component of a pair and $\{x|C(x)\}$ represents a set of all x that satisfy the condition C. The above approach has been successfully used to formalize and verify both discrete [14,13] and continuous random variables [11] in HOL.

Expectation theory plays a vital role in the domain of probabilistic analysis as it is a lot easier to judge performance issues based on the average value of a random variable, which is a single number, rather than its distribution function. Building on the above mentioned probabilistic analysis infrastructure, [12] presents a higher-order-logic definition of expectation for discrete random variables. This function has been used to successfully verify the average values of most of the commonly used discrete random variables. For example, [13] presents the verification of average value of the Binomial random variable, which will be later utilized in this paper for memory array analysis.

Lemma 1: *Expectation of Binomial(m,p) Random Variable*

$$\vdash \forall\ \texttt{m p. } 0 \leq \texttt{p} \wedge \texttt{p} \leq 1 \Rightarrow \texttt{expec (λs. prob\_bino m p s) = m p}$$

where (λ`x.t`) represents a lambda abstraction function in HOL that maps its argument x to $t(x)$ and `prob_bino` is the HOL function for the Binomial random variable modeled using the above mentioned approach.

The higher-order-logic probabilistic analysis approach was further strengthened by some additional formalization related to expectation theory in [10]. This includes a formal definition of the variance characteristic, which is used for measuring dispersion of a random variable. This definition of variance can be utilized to verify variance characteristics of most of the commonly used discrete random variables, e.g., [13] presents the verification of variance of the Binomial random variable as the following theorem.

Lemma 2: *Variance of Binomial(m,p) Random Variable*

$$\vdash \forall\ \texttt{m p.}\ 0 \leq \texttt{p} \wedge \texttt{p} \leq 1 \Rightarrow \texttt{variance}\ (\lambda \texttt{s. prob\_bino m p s}) = \texttt{m p}\ (1 - \texttt{p})$$

The work in [13], also includes the verification of some classical properties of expectation and variance in HOL. One such property is the Chebyshev's inequality, which plays a vital role in verifying tail distribution bounds of probabilistic systems within the HOL theorem prover and is given below

Lemma 3: *Chebyshev's Inequality*

$$\begin{aligned}&\vdash \forall\ \texttt{R a.}\ (0 < \texttt{a}) \wedge (0 < \texttt{variance R}) \wedge \\ &\quad (\texttt{summable}(\lambda \texttt{n.}\ \texttt{n}\ \mathbb{P}\{\texttt{s} \mid \texttt{fst (R s) = n}\})) \wedge \\ &\quad (\texttt{summable}(\lambda \texttt{n.}\ \texttt{n}^2\ \mathbb{P}\{\texttt{s} \mid \texttt{fst (R s) = n}\})) \\ &\Rightarrow \mathbb{P}\ \{\texttt{s} \mid \texttt{abs (fst (R s) - expec R)} \geq \texttt{a}\} \leq \frac{\texttt{variance R}}{\texttt{a}^2}\end{aligned}$$

where the HOL predicate `summable` is $True$ if the infinite summation of its *real* sequence argument exists [9], i.e., $\exists x.\ \lim_{k \to \infty} \sum_{n=0}^{k} f(n) = x$. Thus, the *summable* assumptions in the above theorem state that the theorem is only valid for a random variable R with well-defined expectation and variance values.

In this paper, we utilize the above mentioned infrastructure for conducting formal probabilistic analysis of reconfigurable memory arrays, a novelty that to the best of our knowledge does not exist in the open literature so far.

4 Formal Stuck-at Fault Memory Model

In this section, we develop a formal generic stuck-at fault model for reconfigurable memory arrays. This model will be used to formally reason about the statistical properties and repairability of the memory arrays in the next section. Our formalization approach is mainly inspired by the analytical model developed in [25] for the paper-and-pencil based analysis of reconfigurable memory arrays.

The reconfigurable memory array can be modeled as a bipartite graph (R, C, F). In this bipartite graph, R represents the set of nodes representing rows of the memory array, C is the set of nodes representing the columns of memory array, and F is a set of edges, with each edge connecting one node in the set R to a node in the set C, and represents a fault in the memory array. It is important to note here that the number of elements in the set F and their identities is a random quantity as fault occurrence is an unpredictable event. Therefore, we associate a probability p with every possible pair combination of the elements of the sets R and C of being included in the set F. Also, the occurrence of stuck-at faults, and thus the inclusion of a pair in the set F, is assumed to be independent and identically distributed in this model.

For illustration purposes, consider a square memory array of size n x n with *sr* spare rows and *sc* spare columns and four stuck-at faults, as shown in Figure 1.(a). The corresponding bipartite graph model of the memory array is given in Figure 1.(b). In this model, each of the four faults is represented as an edge connecting a row and column node.

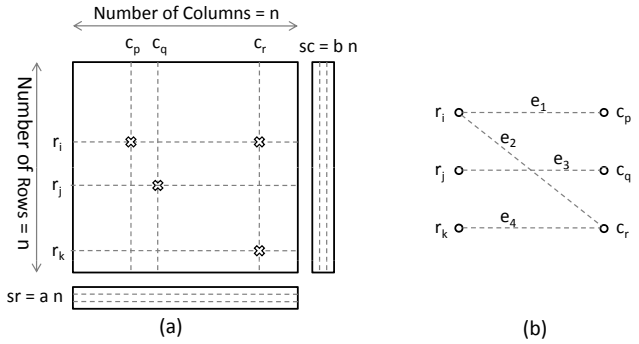

Fig. 1. Memory Array Model

A stuck-at fault occurring at location (x, y) in the memory array can be repaired by replacing either row x or column y with a spare row or a spare column. Thus, in the worst case scenario when we require one row or column to repair a single fault only, a memory array is considered to be absolutely repairable if its total number of stuck-at faults is less than the available number of spare columns or rows. This repairability problem is similar to the vertex cover problem of the bipartite graph and is known to be an NP complete problem [16]. Therefore, we consider solutions to this problem using probability theory and define the probability of repairability, using our memory array model, as follows

$$\Pr(|F| \leq sr + sc) \tag{1}$$

where Pr and $|F|$ represent the probability function and cardinality of a set F, respectively. Equation (1) represents the probability of the event when the number of stuck-at faults $|F|$, a random quantity, is less than the total number of spare rows and columns $sr + sc$. We can express Equation (1) in terms of the number of rows or columns of a square n x n reconfigurable memory array as

$$\Pr(|F| \leq (a + b)n) \tag{2}$$

where $a = \frac{sr}{n}$ and $b = \frac{sc}{n}$. The values of a and b are bounded in the real interval $[0, 1]$, since the number of spare rows and spare columns is usually a small fraction of the total number of rows and columns in the array and can never exceed it.

In this paper, our primary goal is to formally verify that if the probability of stuck-at fault occurrence is given by the following expression

$$p = \frac{(a + b)}{n} - \frac{w(n)}{n\sqrt{n}} \tag{3}$$

then the memory array is almost always repairable, whereas $w(n) \rightarrow \infty$ as $n \rightarrow \infty$. The term almost always repairable in the above context means that the probability of repairability tends to 1 as n becomes very very large. The above expression for the stuck-at fault occurrence probability has been initially

proposed and analyzed using informal techniques in [25]. Our contribution in this paper is to formally verify the above argument using the HOL theorem prover.

We proceed in this direction by first modeling the number of faults or the cardinality of the set F using the following higher-order-logic functions.

Definition 1: *Stuck-at Fault Memory Model*

```
⊢ (∀ p. mem_fault_model_helper 0 p = unit 0) ∧
   ∀ c p. mem_fault_model_helper (c + 1) p =
     bind (mem_fault_model_helper c p)
  (λa. bind (prob_bern p) (λb. unit (if b then (a+1) else a)))

⊢ (∀ c p. mem_fault_model 0 c p = unit 0) ∧
   ∀ r c p. mem_fault_model (r + 1) c p =
     bind (mem_fault_model r c p)
    (λa. bind (mem_fault_model_helper c p) (λb. unit (a + b)))
```

The function `mem_fault_model` accepts three parameters: the cardinalities of the sets R and C and the probability of fault occurrence p. It recursively manipulates these three parameters, with the help of the function `mem_fault_model_helper` and returns the number of faults found in the memory array of size $|R|$ x $|C|$. It is important to note that the fault occurrence behavior, which is the random component in this model, is represented by the formalized Bernoulli random variable function `prob_bern` [14] above. The function `mem_fault_model` basically performs a Bernoulli trail, with the probability of obtaining a $True$ being equal to the probability of fault occurrence, for each cell of the memory array and returns the total number of $True$ outcomes obtained.

Now, in order to verify the condition of repairability, given in Equation (3), we define the following special case of our general memory model.

Definition 2: *Stuck-at Fault Memory Model for Repairability Problem*

```
⊢ ∀ n a b w. mem_fault_model_rep n a b w =
    mem_fault_model n n ((a+b)/n − w(n)/(n√n))
```

The function `mem_fault_model_rep` accepts four parameters: the cardinality of the sets R and C of a square reconfigurable memory array as a *natural* number n, the fractions of spare rows and columns as *real* numbers a and b, respectively, and the *real* sequence w with data type $(natural \rightarrow real)$. It utilizes the function `mem_fault_model`, given in Definition 1, to return the number of stuck-at faults for the specific case of a square n x n memory array with the fault occurrence probability equal to the expression, given in Equation (3).

For simplifying the interactive proofs related to the function `mem_fault_model_rep`, it can be alternately expressed as follows

Lemma 4: *Alternate Stuck-at Fault Memory Model for Repairability Problem*

```
⊢ ∀ n a b w.
   mem_fault_model_rep n a b w = prob_bino n² ((a+b)/n − w(n)/(n√n))
```

The proof of the above lemma is primarily based on the fact that the stuck-at fault occurrences in our memory array model are independent and identically distributed and so are the Bernoulli random variables used in the function definition of `mem_fault_model_rep`. This allows us to express the summation of n^2 Bernoulli(p) random variables in the function `mem_fault_model_rep` as a Binomial(n^2, p) random variable, since Binomial(m, p) random variable basically counts the number of successes in m independent and identically distributed Bernoulli trials, with a success probability p [4].

5 Statistical Properties and Repairability Condition

In this section, we utilize the function `mem_fault_model_rep` to formally verify a couple of statistical properties regarding the number of faults and the almost always repairability condition for an n x n reconfigurable memory array with stuck-at fault occurrence probability given by Equation (3). These verification results play a vital role in designing reliable reconfigurable memory arrays.

5.1 Average Number of Stuck-at Faults

With the probability of stuck-fault occurrence, given by Equation (3), the average number of stuck-at faults for an n x n memory array is given by

$$Ex[|F|] = n^2\left(\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}}\right) \tag{4}$$

This property can be formally expressed in higher-order logic using our formal definition of the number of faults, given in Definition 2, as follows.

Theorem 1: *Average Number of Stuck-at Faults*

$$\begin{aligned}&\vdash \forall\ \mathtt{a\ b\ n\ w.}\\ &\quad \mathtt{(0 \le a) \wedge (a \le 1) \wedge (0 \le b) \wedge (b \le 1) \wedge (1 < n) \wedge}\\ &\quad \mathtt{(\forall\ n.\ (0 < w(n)) \wedge (w(n) < (a+b)\sqrt{n}))}\\ &\Rightarrow \mathtt{expec\ (\lambda s.\ mem\_fault\_model\_rep\ n\ a\ b\ w\ s) = \ n^2\left(\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}}\right)}\end{aligned}$$

The first four assumptions in the above theorem ensure that the fractions a and b are bounded by the interval $[0, 1]$ as described in the previous section. Whereas, the precondition $1 < n$ has been used in order to ensure that the given memory array has more than one cell. The last assumption is about the real sequence w and basically provides its upper and lower bounds. These bounds have been used in order to prevent the stuck-at fault occurrence probability p, given in Equation (3), from falling outside its allowed interval $[0, 1]$. It is interesting to note that no such restriction on the sequence w was imposed in the paper-and-pencil based analysis of the repairability problem given in [25]. This fact clearly demonstrates the strength of formal methods based analysis as it allowed us to highlight this corner case, which if ignored could lead to the invalidation

of the whole repairability analysis. The conclusion of Theorem 1 presents the mathematical relation given in Equation (4).

The HOL proof for Theorem 1 is based on Lemma 4 and the expectation relation for the Binomial random variable, given in Lemma 1. The proof involves some arithmetic reasoning to verify that the probability p, given in Equation (3), lies in the interval $[0, 1]$, which is a precondition for Lemma 1.

5.2 Variance of the Number of Stuck-at Faults

The variance of the number of stuck-at faults for an n x n memory array, with the probability of stuck-at fault occurrence given by Equation (3), is given by

$$Var[|F|] = n^2(\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}})(1 - (\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}})) \tag{5}$$

This property can be formally expressed in HOL as follows

Theorem 2: *Variance of the Number of Stuck-at Faults*

$$\begin{aligned}
&\vdash \forall\ \mathtt{a\ b\ n\ w\ s.}\\
&\quad \mathtt{(0 \le a) \wedge (a \le 1) \wedge (0 \le b) \wedge (b \le 1) \wedge (1 < n) \wedge}\\
&\quad \mathtt{(\forall\ n.\ (0 < w(n)) \wedge (w(n) < (a+b)\sqrt{n}))}\\
&\Rightarrow \mathtt{variance}\\
&\quad \mathtt{(\lambda s.\ mem_fault_model_rep\ n\ a\ b\ w\ s) =}\\
&\qquad\qquad \mathtt{n^2(\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}})(1 - (\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}}))}
\end{aligned}$$

and verified using Lemma 2 just like Lemma 1 was used to verify Theorem 1.

5.3 Tail Distribution Bound for the Number of Stuck-at Faults

A tail distribution bound of the number of stuck-at faults for our n x n memory array, with the probability of stuck-at fault occurrence, given by Equation (3), can be expressed as follows.

$$Pr(|F| \le (a+b)n) \ge 1 - \frac{n^2(\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}})(1 - (\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}}))}{n(w(n))^2} \tag{6}$$

Whereas, the corresponding HOL theorem is as follows.

Theorem 3: *Tail Distribution Bound for the number of Stuck-at Faults*

$$\begin{aligned}
&\vdash \forall\ \mathtt{a\ b\ n\ w\ s.}\\
&\quad \mathtt{(0 \le a) \wedge (a \le 1) \wedge (0 \le b) \wedge (b \le 1) \wedge (1 < n) \wedge}\\
&\quad \mathtt{(\forall\ n.\ (0 < w(n)) \wedge (w(n) < (a+b)\sqrt{n}))}\\
&\Rightarrow \mathtt{(\mathbb{P}\ \{s \mid (fst\ (mem_fault_model_rep\ n\ a\ b\ w\ s)) \le (a + b)n\} \ge}\\
&\quad \mathtt{1 - (\frac{n^2(\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}})(1-(\frac{(a+b)}{n} - \frac{w(n)}{n\sqrt{n}}))}{n(w(n))^2})}
\end{aligned}$$

We proceed with the verification of this theorem by splitting its proof goal into two subgoals using the less-than-or-equal-to transitive property as follows.

$$\mathbb{P}\ \{\mathtt{s}\ |\ (\mathtt{fst}\ (\mathtt{prob\_bino}\ \mathtt{n}^2\ (\tfrac{(\mathtt{a+b})}{\mathtt{n}} - \tfrac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})\ \mathtt{s}) > (\mathtt{a}+\mathtt{b})\mathtt{n} - 2\sqrt{\mathtt{n}}\mathtt{w(n)}) \wedge (\mathtt{fst}\ (\mathtt{prob\_bino}\ \mathtt{n}^2\ (\tfrac{(\mathtt{a+b})}{\mathtt{n}} - \tfrac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})\ \mathtt{s}) < (\mathtt{a}+\mathtt{b})\mathtt{n}\ \}$$
$$\leq \mathbb{P}\ \{\mathtt{s}\ |\ (\mathtt{fst}\ (\mathtt{prob\_bino}\ \mathtt{n}^2\ (\tfrac{(\mathtt{a+b})}{\mathtt{n}} - \tfrac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})\ \mathtt{s})) \leq (\mathtt{a}+\mathtt{b})\mathtt{n}\}$$

$$1 - \frac{\mathtt{n}^2(\frac{(\mathtt{a+b})}{\mathtt{n}} - \frac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})(1-(\frac{(\mathtt{a+b})}{\mathtt{n}} - \frac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}}))}{(\mathtt{nw(n)w(n)})} \leq$$
$$\mathbb{P}\ \{\mathtt{s}\ |\ (\mathtt{fst}\ (\mathtt{prob\_bino}\ \mathtt{n}^2\ (\tfrac{(\mathtt{a+b})}{\mathtt{n}} - \tfrac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})\ \mathtt{s})) > (\mathtt{a}+\mathtt{b})\mathtt{n} - 2\sqrt{\mathtt{n}}\mathtt{w(n)} \wedge (\mathtt{fst}\ (\mathtt{prob\_bino}\ \mathtt{n}^2\ (\tfrac{(\mathtt{a+b})}{\mathtt{n}} - \tfrac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})\ \mathtt{s})) < (\mathtt{a}+\mathtt{b})\mathtt{n}\ \}$$

The first subgoal can be verified using the basic probability axiom ($\forall A\ B.A \subseteq B \Rightarrow (Pr(A) \leq Pr(B))$) since the set on the left-hand-side (LHS) of the inequality is a subset of the set on the right-hand-side (RHS). Whereas, by rewriting the two inequalities in the argument of the probability function of subgoal 2 using absolute value theorem ($((|y-x| < d) = (x-d < y < x+d)$ we get:

$$1 - \frac{\mathtt{n}^2(\frac{(\mathtt{a+b})}{\mathtt{n}} - \frac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})(1-(\frac{(\mathtt{a+b})}{\mathtt{n}} - \frac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}}))}{(\mathtt{nw(n)w(n)})} \leq$$
$$\mathbb{P}\ \{\ \mathtt{s}\ |\ |\mathtt{fst}\ (\mathtt{prob\_bino}\ \mathtt{n}^2\ (\tfrac{(\mathtt{a+b})}{\mathtt{n}} - \tfrac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})\ \mathtt{s}) - ((\mathtt{a}+\mathtt{b})\mathtt{n} - \sqrt{\mathtt{n}}\mathtt{w(n)})| < \sqrt{\mathtt{n}}\mathtt{w(n)}\ \}$$

Now using the complement probability law ($\forall A.Pr(\bar{A}) = 1 - Pr(A)$) along with Theorems 1 and 2, we can rewrite the above sub goal as follows

$$\mathbb{P}\ \{\mathtt{s}\ |\ |\mathtt{fst}\ (\mathtt{prob\_bino}\ \mathtt{n}^2\ (\tfrac{(\mathtt{a+b})}{\mathtt{n}} - \tfrac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})\ \mathtt{s}) - \mathtt{expec}\ (\lambda \mathtt{s.}\ \mathtt{prob\_bino}\ \mathtt{n}^2\ (\tfrac{(\mathtt{a+b})}{\mathtt{n}} - \tfrac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})\ \mathtt{s})| \geq \sqrt{\mathtt{n}}\mathtt{w(n)}\ \} \leq \frac{\mathtt{variance}(\lambda \mathtt{s.prob\_bino}\ \mathtt{n}^2(\frac{(\mathtt{a+b})}{\mathtt{n}} - \frac{\mathtt{w(n)}}{\mathtt{n}\sqrt{\mathtt{n}}})\mathtt{s})}{(\sqrt{\mathtt{n}}\mathtt{w(n)})^2}$$

The above subgoal can now be discharged from the HOL goal stack by using Chebyshev's inequality, given in Lemma 3, along with some arithmetic reasoning.

5.4 Repairability Problem

Now, we use the statistical properties verified so far to analyze the repairability problem, i.e., an n x n reconfigurable memory array with the probability of stuck-at fault occurrence given by Equation (3), is almost always repairable.

$$\lim_{n\to\infty} \Pr(|F| \leq (a+b)n) = 1 \tag{7}$$

The corresponding HOL theorem is as follows

Theorem 4: *Repairability Problem of Stuck-at Faults*

$$\vdash \forall\ \mathtt{a\ b\ w.}\ (\mathtt{0} \leq \mathtt{a}) \wedge (\mathtt{a} \leq \mathtt{1}) \wedge (\mathtt{0} \leq \mathtt{b}) \wedge (\mathtt{b} \leq \mathtt{1}) \wedge (\forall\ \mathtt{n.}\ (\mathtt{0} < \mathtt{w(n)}) \wedge (\mathtt{w(n)} < (\mathtt{a}+\mathtt{b})\sqrt{\mathtt{n}})) \wedge (\mathtt{lim}\ (\lambda \mathtt{n.}\ \tfrac{\mathtt{1}}{\mathtt{w(n)}}) = \mathtt{0}) \Rightarrow (\mathtt{lim}\ (\lambda \mathtt{n.}\ \mathbb{P}\ \{\mathtt{s}\ |\ (\mathtt{fst}\ (\mathtt{num\_of\_faults\ n\ a\ b\ w\ s})) \leq (\mathtt{a}+\mathtt{b})\ \mathtt{n}\}) = 1))$$

where `lim M` represents the HOL formalization of the limit of a real sequence M (i.e., $lim\ M = \lim_{n\to\infty} M(n)$) [9]. The new assumption $\texttt{(lim}(\lambda\texttt{n.}\frac{\texttt{1}}{\texttt{w(n)}}\texttt{)} = \texttt{0)}$ formally represents the intrinsic characteristic of *real* sequence w that it tends to infinity as its *natural* argument becomes very very large.

We proceed with the verification of Theorem 4 by first splitting its proof goal into the following two subgoals, based on some simple arithmetic reasoning.

$$\texttt{lim}(\lambda\texttt{n. } \mathbb{P}\{\texttt{s | fst (mem\_fault\_model\_rep n a b w s)} \leq \texttt{(a + b)n}\}) \leq 1$$

$$1 \leq \texttt{lim}(\lambda\texttt{n. } \mathbb{P}\{\texttt{s | fst (mem\_fault\_model\_rep n a b w s)} \leq \texttt{(a + b)n}\})$$

The first subgoal can be verified using the basic probability axiom ($\forall A.Pr(A) \leq 1$). Whereas, we utilize Theorem 3 and the transitivity property of less-than-or-equal-to for real numbers to rewrite the second subgoal as follows.

$$\texttt{1} \leq \texttt{lim } (\lambda\texttt{n. 1 - } \frac{\texttt{n}^2(\frac{\texttt{a+b}}{\texttt{n}} - \frac{\texttt{w(n)}}{\texttt{n}\sqrt{\texttt{n}}})(1 - \frac{\texttt{a+b}}{\texttt{n}} + \frac{\texttt{w(n)}}{\texttt{n}\sqrt{\texttt{n}}})}{\texttt{n(w(n))}^2})$$

The expression in the RHS of the above inequality can be rewritten as follows using some arithmetic reasoning.

$$\texttt{1} \leq \texttt{lim } (\lambda\texttt{n. 1 - } ((\frac{\texttt{a+b}}{\texttt{w(n)}} - \frac{\texttt{1}}{\sqrt{\texttt{n}}})(\frac{\texttt{1}}{\texttt{w(n)}} - \frac{\texttt{a+b}}{\texttt{nw(n)}} + \frac{\texttt{1}}{\texttt{n}\sqrt{\texttt{n}}})))$$

This subgoal can now be verified as the limit value of the expression on the RHS tends to 1, since all the denominator terms in this expression tend to ∞ as n becomes very very large. This also concludes the proof for Theorem 4.

Our results clearly demonstrate the effectiveness of the theorem proving based reconfigurable memory array analysis approach. Due to the formal nature of the model and inherent soundness of theorem proving, we have been able to verify the properties of interest regarding the given memory array with 100% precision; a novelty which is not available in simulation. Similarly, due to the high expressibility of higher-order logic we have been able to formally reason about statistical properties of the problem that cannot be analyzed using a probabilistic model checker. The proposed approach is also superior than the paper-and-pencil proof methods in terms of accuracy. In the paper-and-pencil approach, the proof checking and associated bookkeeping is an error prone process, specially when dealing with large proofs, and thus often leads either to wasted time and effort or a wrong result. On the other hand, in theorem proving, these complicated tasks are done by the computer within a sound core, which is based on a very few axioms and inference rules. Each proven theorem can be logically traced back to these basic axioms and the associated proof steps can be linked to the basic inference rules. Due to this inherent soundness, it is impossible to prove wrong statements in a theorem prover.

The above mentioned additional benefits, associated with the theorem proving approach, are attained at the cost of the time and effort spent, while formalizing the memory array and formally reasoning about its properties, by the user. But, the fact that we were building on top of already verified probability theory related results helped significantly in this regard as this analysis only consumed approximately 80 man-hours and 1200 lines of HOL code by an expert user.

6 Conclusions

In this paper, we utilized the mathematical probability theory formalized in a higher-order-logic theorem prover to analyze reconfigurable memory arrays in the presence of stuck-at faults. To the best of our knowledge, this is the first study on using these kind of techniques for such an application. We developed a higher-order-logic based formal stuck-at fault model for reconfigurable memory arrays, and based on this model we formally verified some key statistical properties and repairability condition. The rigorous exercise of developing a computer based formal model for the memory array and analyzing it using mechanized mathematical reasoning allowed us to discover a couple of critical assumptions that are missed by almost all of the paper-and-pencil based analysis, that we came across, of a similar problem. Due to the formal nature of the models and the inherent soundness of theorem proving systems, the analysis is guaranteed to provide exact answers. These feature makes the proposed approach very useful for the probabilistic analysis of memory arrays that are to be used in safety critical and highly sensitive areas.

Our approach for the probabilistic analysis of stuck-at faults in memory reconfigurable arrays is quite general and can be extended and easily adapted to conduct precise probabilistic analysis of other kinds of fault models, like transition faults, coupling faults, and neighborhood pattern sensitive faults, as well. The random or unpredictable elements found in these models can be represented using an appropriate random variable from the existing library of formalized discrete [14,12,13] and continuous random variables [11], and the precise statistical quantities associated with the parameters of interest may then be verified within the sound core of a higher-order-logic theorem prover. For example, the probabilistic analysis approach for coupling faults [25] can be adapted in a theorem prove using the formal definition of the Binomial random variable along with the theorems regarding its expectation and variance. Similarly, other statistical properties, such as the conditions for irrepairability and tail distribution bounds based on Markov's inequality, can also be verified.

References

1. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model Checking Algorithms for Continuous time Markov Chains. IEEE Transactions on Software Engineering 29(4), 524–541 (2003)
2. Blough, D.M.: Performance Evaluation of a Reconfiguration-Algorithm for Memory Arrays containing Clustered Faults. IEEE Transactions on Reliability 45(2), 274–284 (1996)
3. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)
4. DeGroot, M.: Probability and Statistics. Addison-Wesley, Reading (1989)
5. Devroye, L.: Non-Uniform Random Variate Generation. Springer, Heidelberg (1986)
6. Gordon, M.J.C.: Mechanizing Programming Logics in Higher-0rder Logic. In: Current Trends in Hardware Verification and Automated Theorem Proving, pp. 387–439. Springer, Heidelberg (1989)

7. Gordon, M.J.C., Melham, T.F.: Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic. Cambridge University Press, Cambridge (1993)
8. Gupta, A.: Formal Hardware Verification Methods: A Survey. Formal Methods in System Design 1(2-3), 151–238 (1992)
9. Harrison, J.: Theorem Proving with the Real Numbers. Springer, Heidelberg (1998)
10. Hasan, O.: Formal Probabilistic Analysis using Theorem Proving. PhD Thesis, Concordia University, Montreal, QC, Canada (2008)
11. Hasan, O., Tahar, S.: Formalization of the Continuous Probability Distributions. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 3–18. Springer, Heidelberg (2007)
12. Hasan, O., Tahar, S.: Verification of Expectation Properties for Discrete Random Variables in HOL. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007. LNCS, vol. 4732, pp. 119–134. Springer, Heidelberg (2007)
13. Hasan, O., Tahar, S.: Formal Verification of Tail Distribution Bounds in the HOL Theorem Prover. Mathematical Methods in the Applied Sciences (2008), http://www3.interscience.wiley.com/journal/120747455/abstract
14. Hurd, J.: Formal Verification of Probabilistic Algorithms. PhD Thesis, University of Cambridge, Cambridge, UK (2002)
15. ITRS (2008), http://www.itrs.net/links/2003itrs/home2003.htm
16. Kuo, S., Fuchs, W.K.: Efficient Spare Allocation for Reconfigurable Arrays. IEEE Design & Test of Computers 4(1), 24–31 (1987)
17. Kwiatkowska, M., Norman, G., Parker, D.: Quantitative Analysis with the Probabilistic Model Checker PRISM. Electronic Notes in Theoretical Computer Science 153(2), 5–31 (2005)
18. Low, C.P., Leong, H.W.: Probabilistic Analysis of Memory Reconfiguration in the Presence of Coupling Faults. In: Proceedings of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems (1992)
19. MacKay, D.J.C.: Introduction to Monte Carlo Methods. In: Learning in Graphical Models, NATO Science Series, pp. 175–204. Kluwer Academic Press, Dordrecht (1998)
20. Miczo, A.: Digital Logic Testing and Simulation. Wiley Interscience, Chichester (2003)
21. Nicolaidis, M., Achouri, N., Anghel, L.: A Diversified Memory Built-in Self-repair Approach for Nanotechnologies. In: Proceedings of the 22nd IEEE VLSI Test Symposium, pp. 313–318 (2004)
22. Rutten, J., Kwaiatkowska, M., Normal, G., Parker, D.: Mathematical Techniques for Analyzing Concurrent and Probabilisitc Systems. CRM Monograph Series, vol. 23. American Mathematical Society (2004)
23. Sehgal, A., Dubey, A., Marinissen, E.J., Wouters, C., Vranken, H., Chakrabarty, K.: Redundancy Modelling and Array Yield Analysis for Repairable Embedded Memories. IEE Proceedings of Computers and Digital Techniques 152(1), 97–106 (2005)
24. Sen, K., Viswanathan, M., Agha, G.: VESTA: A Statistical Model-Checker and Analyzer for Probabilistic Systems. In: Proc. IEEE International Conference on the Quantitative Evaluation of Systems, pp. 251–252 (2005)
25. Shi, W., Fuchs, W.K.: Probabilistic Analysis and Algorithms for Reconfiguration of Memory Arrays. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 11(9), 1153–1160 (1992)