A Modular Memory BIST for Optimized Memory Repair
Philipp Öhler, Alberto Bosio, Giorgio Di Natale, Sybille Hellebrand

To cite this version:

HAL Id: lirmm-00363724
https://hal-lirmm.ccsd.cnrs.fr/lirmm-00363724
Submitted on 24 Feb 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Modular Memory BIST for Optimized Memory Repair

Philipp Öhler¹, Alberto Bosio², Giorgio Di Natale², and Sybille Hellebrand¹
¹University of Paderborn, Germany
²LIRMM, University of Montpellier, France
{oehler, sybille.hellebrand}@upb.de, {alberto.bosio, giorgio.dinatale}@lirmm.fr

Abstract

An efficient on-chip infrastructure for memory test and repair is crucial to enhance yield and availability of SoCs. Most of the existing built-in self-repair solutions reuse IP-Cores for BIST without modifications. However, this prevents an optimized test and repair interaction. In this paper, the concept of modular BIST for memories is introduced, which supports a more efficient interleaving of test and repair and can be achieved with only small modifications in the BIST control.

1. Introduction

Today system-on-chips (SoC) embed several hundreds of different memory cores occupying more than 90% of the SoC chip area. The yield of the entire system is therefore dominated by the memory yield. Memory test and repair capabilities are provided to check the functionality of the memory cores and to increase yield. In the presence of manufacturing defects, redundant elements can replace the failing parts of the memory array [1].

Various built-in self-repair (BISR) schemes have been developed [2, 3, 4]. In most of these schemes test and repair coexist. Meanwhile, implementing a memory BIST is possible by using intellectual property (IP) cores or by relying on automated generation flows of CAD tools. Memory repair requires the exact failure information in the cell array, and most repair schemes are flexible in the sense that any march-like test can be used for failure retrieval. Thus the memory BIST cores can be integrated into a test and repair infrastructure without any modification.

The integrated test and repair approach presented in [4] supports an optimal built-in self-repair for memories with redundant rows and columns (“2D redundancy”). The scheme interleaves test and repair analysis to avoid large failure bitmaps. Moreover, it follows a depth first strategy for traversing the binary tree for spare allocation. This supports a hardware implementation scaling well with the number of spares. However, backtracking in the search tree requires a restart of the complete test. A more detailed analysis shows that repeating the complete test may lead to an unnecessarily high increase in test time. To overcome this problem, the concept of a modular march test is introduced in this paper.

2. The Modular Test Strategy

The interleaved test and repair scheme in [4] makes repair decisions as soon as faults are detected during test. The nodes in the search tree for the best repair configuration correspond to detected faults while the edges represent repair decisions (row or column). Backtracking is necessary when a chosen path cannot provide a solution or to prove that an already found solution is optimal. In Figure 1 an example is given.

![Figure 1: Partial search tree during repair analysis](image)

The information attached to node $x$ shows that a fault at address $a(x)$ has been detected, and the edge $(x, y)$ indicates a row repair. If backtracking to node $x$ occurs, then the test is restarted to determine the remaining faults for the alternative repair decision using a column. This avoids the need for large failure bitmaps, but a complete restart after each backtrack implies a high time penalty.

If a march test with march elements $M_0, ..., M_n$ is used, then the information during which march element a fault has been detected can help to reduce the overall test and repair time. In Figure 1, the fault at node $x$ has been detected during march element $M_i$. Part of this research has been performed within the framework of the DFG grant DIADEM (HE 1686/2-1).
and this suggests to restart the test after the detection of the fault at address $d(x)$. To exploit this idea properly, it must be taken into account that fault detection in march tests usually requires initialization and sensitization in earlier march elements [5]. As the initialization and sensitization conditions may have been overwritten during the previous search, the respective march elements have to be repeated, too. If initialization starts with march element $M_k$, $k < i$, then it is even sufficient to work with a “lightweight” version $M_k^*$ of $M_k$, which only contains the memory operations for initialization. $M_k^*$ can easily be compiled from the fault detection profiles of the algorithms and the considered fault models. The hardware implementation requires only small modifications of the test and repair control.

3. Experiments and Results

The same experimental set-up as in [4] has been used to analyze two different fault sets: set(a), which contains stuck-at faults, and set(b), which consists of stuck-at faults, transition faults, and coupling faults. Accordingly, two different march tests have been applied: the March X algorithm for set(a) and the March C- algorithm for set(b) [5]. The results are shown in Table 1.

<table>
<thead>
<tr>
<th>Defects</th>
<th>Test restarts</th>
<th>#OP original</th>
<th>#OP modular</th>
<th>Modular/Original</th>
<th>Test restarts</th>
<th>#OP original</th>
<th>#OP modular</th>
<th>Modular/Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.183</td>
<td>6.650</td>
<td>6.442</td>
<td>0.969</td>
<td>2.163</td>
<td>10.446</td>
<td>10.446</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>3.601</td>
<td>16.370</td>
<td>13.842</td>
<td>0.846</td>
<td>3.601</td>
<td>21.458</td>
<td>21.320</td>
<td>0.994</td>
</tr>
<tr>
<td>3</td>
<td>9.627</td>
<td>40.740</td>
<td>30.358</td>
<td>0.745</td>
<td>9.627</td>
<td>44.238</td>
<td>40.114</td>
<td>0.907</td>
</tr>
<tr>
<td>4</td>
<td>16.364</td>
<td>67.518</td>
<td>49.636</td>
<td>0.735</td>
<td>16.364</td>
<td>87.788</td>
<td>76.260</td>
<td>0.869</td>
</tr>
<tr>
<td>5</td>
<td>18.065</td>
<td>74.340</td>
<td>48.068</td>
<td>0.647</td>
<td>18.065</td>
<td>134.012</td>
<td>84.920</td>
<td>0.634</td>
</tr>
<tr>
<td>6</td>
<td>32.307</td>
<td>126.796</td>
<td>84.114</td>
<td>0.663</td>
<td>32.307</td>
<td>210.000</td>
<td>142.136</td>
<td>0.677</td>
</tr>
<tr>
<td>7</td>
<td>49.278</td>
<td>187.816</td>
<td>129.500</td>
<td>0.690</td>
<td>49.278</td>
<td>287.560</td>
<td>200.927</td>
<td>0.699</td>
</tr>
<tr>
<td>8</td>
<td>162.830</td>
<td>235.708</td>
<td>157.808</td>
<td>0.670</td>
<td>162.830</td>
<td>372.054</td>
<td>264.211</td>
<td>0.710</td>
</tr>
<tr>
<td>9</td>
<td>76.926</td>
<td>273.414</td>
<td>195.170</td>
<td>0.714</td>
<td>76.926</td>
<td>485.338</td>
<td>341.234</td>
<td>0.703</td>
</tr>
<tr>
<td>10</td>
<td>75.073</td>
<td>292.866</td>
<td>176.178</td>
<td>0.602</td>
<td>75.073</td>
<td>630.338</td>
<td>344.206</td>
<td>0.546</td>
</tr>
<tr>
<td>11</td>
<td>68.164</td>
<td>242.246</td>
<td>178.316</td>
<td>0.736</td>
<td>68.164</td>
<td>478.016</td>
<td>345.096</td>
<td>0.722</td>
</tr>
<tr>
<td>12</td>
<td>57.454</td>
<td>212.256</td>
<td>159.666</td>
<td>0.752</td>
<td>57.454</td>
<td>390.848</td>
<td>304.760</td>
<td>0.780</td>
</tr>
</tbody>
</table>

The results are interesting in two ways. Firstly, for a small number of defects when only a few restarts are needed, the modular strategy provides only a small improvement. For a larger number of defects, when the search gets more complex, the number of exercised memory operations is reduced to approximately 70% of the original solution in [4]. Secondly, when the test algorithm gets more complex, then the achievement is even better in most cases.

4. Conclusions

In this paper, modular memory testing has been introduced as a strategy for an improved test and repair interaction. Combined with the integrated test and repair approach described in [4], this strategy helps to considerably reduce the time penalty caused by backtracks in the search tree for spare allocation.

5. References