The Influence of Implementation Technology on Dependability Parameters

Jan Schmidt, Petr Fišer, Jiří Baláček
Czech Technical University in Prague, Thákurova 9, 160 00 Prague, Czech Republic

Abstract

Circuits which are designed to be dependable are evaluated after gate-level design. To demonstrate the influence of implementation technology on computed dependability parameters, we developed a simple method which transforms the evaluation problem into conceptual hardware and then to SAT instances. The method can accommodate any combinational fault model. The performed evaluation demonstrated that the dependability parameters of the implementations correlate to a significant degree.

Keywords: dependability, SAT, fault classification, generalized miter

1. Introduction

The principal way to improve the dependability of a circuit is to introduce redundancy. One possible strategy is to detect errors at output signals and take appropriate measures during circuit operation. This technique is called Concurrent Error Detection (CED, [1]).

Redundancy must be introduced with care, as redundant blocks are also prone to faults, and the point of diminishing return can be reached easily. Numerous schemes were devised to balance redundancy and dependability. To evaluate variant designs, we need to know how much dependability we get for a given investment into redundant circuits. Initially, dependability meant roughly what is today called robustness. In this paper, we adhered to the original meaning from [2], where dependability parameters were introduced to quantify dependability.

The standard design flow is to design the circuit first, to construct its redundancy afterward, and then to evaluate its dependability parameters. The underlying assumption is that the actual design, the technology used and its resulting fault models influence the dependability parameters to a large degree. In many studies, the ubiquitous stuck-at (S@) fault models were used.

Recent Automatic Test Pattern Generation (ATPG) programs [3] and procedures based on solving the Satisfiability Problem (SAT) [4] permit analysis with a variety of fault models suitable for a particular circuit implementation technology. This in turn enables us to see the influence of technology on dependability. In other words, we ask whether there are circuits hard to make dependable or technologies hard to make dependable.

To study this question, we needed a simple framework. Recently, two approaches to robustness analysis and other tasks dealing with faults exist. The first one, represented by [5] and [6], transforms the task instance to an ATPG task instance. The ATPG program then may or may not convert it internally to one or more SAT instances [7], [8]. The other approach, most notably represented by [4] and [9], converts an instance of the task to conceptual hardware (hardware which is not intended for synthesis) and then constructs SAT instances from that hardware.

Both approaches can be seen as special cases of a more general method, which can be summarized as follows. Firstly, transform the task instance into a piece of conceptual hardware together with assertions about the hardware. Secondly, use formal verification methods to prove or disprove the assertions (e.g. [3], [10]). If required, transform the assertions into conceptual hardware as well [11]. Thirdly, transform the answers back to the answers to the original task instance.

This is a powerful framework, which can even produce answers about sequential behavior in the presence of multiple faults [4], [9], [12]. For our study, combinational circuits (or full-scan circuits) were sufficient. Furthermore, only fault classification was required, without the need to analyze multiple fault impact.

Therefore, we present a simple framework for this limited situation, which constructs conceptual hardware representing the circuit and the assertions directly. We borrowed the term miter from ATPG [7], although the term monitor from [8] and other sources has a similar meaning.

We present a method to determine the value of an arbitrary Boolean formula over input vectors, error-free output vectors, and error-stricken output vectors of a combinational circuit. The formula can be quantified over all input vectors or their subset. We demonstrate our method to be compatible with methods used for multiple fault and sequential circuit modeling.

Although we cannot overcome the exponential worst case complexity of the SAT problem, we have practical solvers for the Boolean Satisfiability Problem, which solve both satisfiable and unsatisfiable instances effectively. The method also benefits from the fact that the SAT instances encountered during ATPG, and, as it was discovered, robustness analysis, are far simpler to
solve than the worst case [10], [13], [14].

We present the method as an extension of SAT ATPG first in its general form. Then we review the class of dependable circuits studied and present their fault classification. We demonstrate application of the proposed method on this problem, and finally show and discuss classification results for two different implementation technologies.

2. Predicate evaluation

2.1. The SAT-Based ATPG

Let the circuit in question realize a Boolean function \( F(x) \) over input \( x \). The circuit has \( n \) primary inputs and \( m \) primary outputs.

\[
\begin{array}{c}
\text{F} \\
n \quad m
\end{array}
\]

Figure 1: A circuit \( F \) with \( n \) inputs and \( m \) outputs

Denote \( F_{fb}(x) \) the Boolean function characterizing the circuit with a given fault. The question whether the fault can be detected is answered by the predicate

\[
\exists x, F(x) \neq F_{fb}(x)
\]  \hspace{1cm} (1)

In SAT-based ATPGs, this is understood as a circuit, see Figure 2. The fault-free and faulty circuits provide \( F(x) \) and \( F_{fb}(x) \), respectively. The predicate itself is also expressed as a circuit called the miter [7]. The characteristic function of the entire circuit is then constructed in Conjunctive Normal Form (CNF) using the Tseitin’s transformation [15] and its satisfiability is solved. If the instance is unsatisfiable, the fault cannot be tested. If it is satisfiable, all solutions are input vectors testing the fault.

\[
\begin{array}{c}
\text{F} \\
x \quad n \quad m \quad F_{fb} \\
m \quad \quad \quad m
\end{array}
\]

Figure 2: Circuit description of the ATPG SAT instance

For details on the SAT-based ATPGs, see [7], [11], [8].

2.2. General Predicates

Let \( x, F(x) \) and \( F_{fb}(x) \) have the same meaning as above. Let

\[
\exists x, G(x, F(x), F_{fb}(x))
\]

be any Boolean predicate over \( x, F(x) \) and \( F_{fb}(x) \). Then \( G \) can also be understood as a circuit, see Figure 3. As it has the same role as in ATPG or model checking, we call it the generalized miter. Its characteristic function can be constructed as in the ATPG case, and the SAT instance is solved.

A universally quantified predicate

\[
\forall x, H(x, F(x), F_{fb}(x))
\]

can simply be converted to

\[
\neg \exists x, \neg H(x, F(x), F_{fb}(x))
\]

The construction of \( \neg H \) might seem difficult. When seen as a circuit, however, it suffices to add an inverter. This causes one more variable and two clauses in the SAT instance [15], which is tolerable. The predicate can be transformed to CNF by other methods as well; the above case illustrates the advantage of seeing it as a circuit. The dependency of the general predicate on \( x \) is useful in situations where not every input vector is admissible. Let \( A \) be the set of admissible input vectors and \( a(x) \) the predicate characterizing the set. Then

\[
\exists x \in A, G(F(x), F_{fb}(x))
\]

becomes

\[
\exists x, a(x) \land G(F(x), F_{fb}(x))
\]

This feature achieves the same effect as the input encoder in [5]. In the case solved there, code generator and detector for the codes in question are comparable in complexity. For other problems, however, to produce a vector may be more difficult than to check that vector.

The technique of generalized mites has a wider utilization than just single fault classification. For the sake of completeness, we demonstrate that it is compatible with procedures used for multiple fault reasoning and analysis of sequential circuits.

2.3. Extension to multiple faults

In [16] and [4], a technique to model multiple faults in a circuit is presented. The presence or absence of each fault is modeled by an associated fault predicate. The actual value of signals in question is modeled by additional primary inputs, which in turn are expressed by free variables in the SAT instance.

The structure of the conceptual hardware including the generalized miter remains the same as in Figure 3, except the faulted copy of \( F \) is controlled by the fault predicates and additional inputs.
2.4. Extension to sequential circuits

[4] uses time-frame unrolling to reason about sequential circuits. The circuit is divided into state register and combinational logic. For each time step, a stage is created comprising a copy of the fault-free and faulty logic. The outputs of both copies in each stage are compared by a miter. The value of state variables passes from stage to stage.

The miter can be replaced by a generalized miter, as in Figure 4. The construction of the generalized miter depends on the way it is specified. If the miter is combinational, then each stage contains one copy of the miter, and all outputs of those stage miters are combined by e.g., an AND gate. If the miter is described as a state machine, then it is unrolled similarly to F and the state of the miter is passed from state to state.

Figure 4: The generalized miter for predicate G in unrolled sequential circuit; x, s, and f are inputs, states, and fault predicates, respectively, in time ti.

3. The Analyzed Architecture

3.1. The Structure of the Dependable Block

The CED strategy proposed in [1], [17] is used in this paper to illustrate principles of the proposed SAT-based predicate evaluation and for the experimental evaluation.

The digital circuit D to be secured by a CED code is supplemented with a predictor P and a checker E, see Figure 5. The predictor can be understood as a copy of the functional circuit together with an encoder. The encoder transforms the vector at the primary outputs of the circuit into the redundancy bits of a selected error detection code. The primary outputs (POs) of the circuit to be secured and the predictor outputs form the code-word whose validity is verified by the checker. The original primary outputs D(x) together with the checker output form the global output F(x), which is m + 1 bits wide.

Any fault in the functional logic D either does not alter the output for a given input vector, or should be detected by the checker. Faults in the predictor and checker either do not affect the operation, or cause false alarms. This architecture can be apprehended as a kind of modification of the well-known duplex scheme [17], [18].

For the purpose of this paper, single parity is used as the error detection code[18]. Thus, the predictor is constructed as a copy of the original circuit supplemented with a XOR tree at its outputs, k = 1 in Figure 5.

The single parity code offers a low area overhead, however its error detection capabilities are limited. Therefore, the fault coverage can also be lower than in the case of the duplex system, and must be analyzed.

Figure 5: Basic concurrent error detection (CED) scheme

3.2. Fault Classification and Dependability Parameters

There are three basic dependability parameters in the field of CED (Concurrent Error Detection) [1], [2]:

**Fault security (FS)** - probability that the erroneous outputs produced for a modeled fault do not belong to the output code-words.

**Self-testing property (ST)** - probability that an input vector occurring during normal operation produces an output vector which does not belong to the code when a modeled fault occurs.

**Totally self-checking (TSC)** - The FS and ST parameters of the circuit are equal to 100%. Totally Self-Checking property offers the highest level of protection.

The faults in the secured block cannot be classified only as detectable or undetectable, as for a common circuit. Their detectability by the checker must also be evaluated [4]. To compute these parameters, an approach based on a fault classification was presented in [4], [19], [18]. The faults are classified into four groups (A, B, C and D) based on their observability on primary outputs of the circuit and detectability by the checker.

**Class A** These faults do not affect the circuit POs for any allowed input vector. This is the class of redundant (undetectable) faults. They have no impact on the FS property, but circuits with these faults cannot be ST.

**Class B** These faults are detectable by at least one input vector and do not produce an incorrect code word (a valid code word, but incorrect) for other input vectors. They have no negative impact on the FS and ST properties, since if such a fault occurs, it is detected by the checker.

**Class C** The faults that produce an incorrect codeword for at least one input vector and cannot be detected by any input vector. This is the class of faults, that can never be detected by the checker and that produce an erroneous output. The circuit with these faults is neither FS nor ST.
Class D these faults cause at least one detectable and one undetectable error on the POs. They are detectable, but also may produce an incorrect output, which is not detected by the checker. The circuit with these faults does not satisfy the FS property.

The FS property can be computed from the number of faults in these classes as:

$$FS = \frac{A + B}{A + B + C + D} \cdot 100\%$$  \hspace{1cm} (2)

The ST property is computed in similar way as:

$$ST = \frac{B + D}{A + B + C + D} \cdot 100\%,$$ \hspace{1cm} (3)

where A, B, C, and D are the numbers of faults in the respective classes.

4. SAT-Based Fault Classification Technique

To apply the SAT-based classification on the above outlined architecture, we must characterize the classes by binary predicates and apply the general scheme form Figure 3.

4.1. Predicates

To compute the dependability parameters of the given architecture, each fault must be classified into one of the classes A, B, C, and D. Four classes need at least two binary predicates to distinguish. In this case, they are easy to derive from the specifications. In principle, the classes are defined by the ability of the fault to cause a detected or an undetected error, which can be formalized as follows:

- $J(x)$ is true iff the input vector $x$ gives an erroneous output $D(x)$ of the faulty circuit and the error is detected ($E(x)$ is true.)

- $K(x)$ is true iff the input vector $x$ gives an erroneous output $D(x)$ of the faulty circuit and the error is not detected ($E(x)$ is false.)

Then the given fault belongs to

- the class A, iff $\neg \exists x, J(x) \land \neg \exists x, K(x)$

- the class B, iff $\exists x, J(x) \land \neg \exists x, K(x)$

- the class C, iff $\neg \exists x, J(x) \land \exists x, K(x)$

- the class D, iff $\exists x, J(x) \land \exists x, K(x)$

Hence, two SAT instances must be solved to classify a fault.

4.2. Generalized Milters

To construct a miter for the $J$ and $K$ predicates, we have to apply the general process leading from the circuit in Figure 1. to the circuit in Figure 3. on the discussed architecture. The output $F(x)$ is in our case decomposed into $D(x)$ and $E(x)$, giving the circuit in Figure 6.

Bringing in the internal structure of $F$ and $F_{fl}$ from Figure 5., we obtain the circuit in Figure 7. The actual predicates apply to all input vectors $x$, therefore $x$ does not enter into the miter circuits. Furthermore, we are interested in faults in the secured circuit $D$ only, not in the predictor or checker, since possible faults there cause just false alarms. These faults do not affect the dependability parameters considered in this paper. Therefore, we can omit $E_{fl}(x)$ from the miters and, therefore, $P_{fl}$ and $E_{fl}$ from the circuit. The final optimized circuit is in Figure 8.

Using $D(x)$, $D_{fl}(x)$ and $E(x)$, we can implement the miters.
Table 2: Total fault counts by category

<table>
<thead>
<tr>
<th>Faults</th>
<th>A</th>
<th>B</th>
<th>NST = C</th>
<th>D</th>
<th>NFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>gates</td>
<td>33251</td>
<td>329</td>
<td>24933</td>
<td>765</td>
<td>7224</td>
</tr>
<tr>
<td>LUTs</td>
<td>61806</td>
<td>9835</td>
<td>45222</td>
<td>1143</td>
<td>5606</td>
</tr>
</tbody>
</table>

\[ J(x) \equiv D(x) \oplus D_{\text{flt}}(x) \land E(x) \]  \hspace{1cm} (4)

\[ K(x) \equiv D(x) \oplus D_{\text{flt}}(x) \land \neg E(x) \]  \hspace{1cm} (5)

5. Experimental Technology Comparison

Using the above described framework, we compared robustness of a set of benchmarks, implemented either structurally (as a network of gates) with S@ faults, or implemented as a set of Look Up Tables (LUTs), considering Single Event Upset (SEU) in the LUT configuration memory as the primary fault mechanism.

The experiments have been performed on 67 ISCAS85 [20], ISCAS89 [21], ITC99 [22] and LGSynth [23] benchmark circuits.

For the S@ faults, the original structural description was used. The fault lists were generated by Atalanta [24] and were free from dominated faults.

The LUT implementations were synthesized by ABC [25] using the command sequence `strash; dch; if; lutpack` as recommended by the authors. Note that this procedure builds the implementation out of LUTS with 4 inputs or less.

The gate implementation was also produced by ABC with the command sequence `strash; dch; map`. The target gate library was the MCNC library [26].

Both implementations of a circuit have obviously different numbers of possible faults. To compare them in a practically relevant manner, we decided to count points of vulnerability, that is, the number of faults which can cause dysfunction of the circuit. The coefficients \( FS \) and \( ST \), which indicate the distance to the Totally Self Checking goal, are of minor importance here. The metrics used were Not Fail Safe

\[ NFS = C + D \]

and Not Self-Testing

\[ NST = C. \]

Table 1 shows the number of faults classified by the above described method.

6. Discussion

The aggregate number of faults by category is given in Table 2, and the statistical properties are summarized in Table 3, using standard correlation and least square linear regression.

6.1. Total number of faults

The total numbers of faults in both technologies are tightly correlated. The circuit itself has the strongest influence. The technology contributes only a constant coefficient, as the regression figure in Table 3 tells us that the LUT implementations has—almost uniformly—twice the number of total faults. The trend is most apparent in large circuits with more faults, as can be seen in Figure 9.

This comparison, however, is influenced by the construction of the fault list for gates. A single S@ fault there can represent more than one dominated fault and hence more than one point of vulnerability, whereas dominance of SEU faults has not been considered.

6.2. A-class faults

The A-class faults are caused by redundancy introduced during synthesis. From the dependability point of view, this is a kind of noise in the implementation. From Figure 10 it can be seen that the values are indeed uncorrelated. 4.5% of the circuits have A-faults only in gate implementation, 57% only in LUT implementation, and 21% in both.

Note that A-class faults might also be introduced by not fully exploiting the 4-LUTs; there are many LUTs having less than 4 actual inputs in the synthesized designs. However, these faults are not considered in our computations, since they are not A-class faults, indeed. If any number of SEUs in the unused parts of LUTs occurs, the dependability is not affected at all.
<table>
<thead>
<tr>
<th>Circuit</th>
<th>Faults</th>
<th>Gates, %</th>
<th>LUTs, SEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>5xpl</td>
<td>422</td>
<td>0</td>
<td>353</td>
</tr>
<tr>
<td>9ynml</td>
<td>446</td>
<td>0 446</td>
<td>0 0 0</td>
</tr>
<tr>
<td>9ym</td>
<td>713</td>
<td>0 713</td>
<td>0 0 0</td>
</tr>
<tr>
<td>a12</td>
<td>400</td>
<td>0 338</td>
<td>8 54 62</td>
</tr>
<tr>
<td>alcom</td>
<td>319</td>
<td>0 291</td>
<td>16 12 28</td>
</tr>
<tr>
<td>alu1</td>
<td>109</td>
<td>0 91</td>
<td>0 18 18</td>
</tr>
<tr>
<td>alu2</td>
<td>1132</td>
<td>117 383</td>
<td>9 623 632</td>
</tr>
<tr>
<td>andm</td>
<td>842</td>
<td>0 632</td>
<td>23 187 210</td>
</tr>
<tr>
<td>b1</td>
<td>37</td>
<td>6 25</td>
<td>0 6 6</td>
</tr>
<tr>
<td>b9</td>
<td>366</td>
<td>2 205</td>
<td>45 114 159</td>
</tr>
<tr>
<td>br1</td>
<td>341</td>
<td>0 230</td>
<td>17 94 111</td>
</tr>
<tr>
<td>br2</td>
<td>296</td>
<td>0 185</td>
<td>20 91 111</td>
</tr>
<tr>
<td>c1355</td>
<td>882</td>
<td>0 704</td>
<td>0 178 178</td>
</tr>
<tr>
<td>c17</td>
<td>22</td>
<td>0 12</td>
<td>0 10 10</td>
</tr>
<tr>
<td>c1908</td>
<td>971</td>
<td>5 437</td>
<td>0 529 529</td>
</tr>
<tr>
<td>c432</td>
<td>553</td>
<td>8 82</td>
<td>0 463 463</td>
</tr>
<tr>
<td>c499</td>
<td>882</td>
<td>0 704</td>
<td>0 178 178</td>
</tr>
<tr>
<td>c8</td>
<td>636</td>
<td>56 489</td>
<td>0 91 91</td>
</tr>
<tr>
<td>cc</td>
<td>219</td>
<td>13 172</td>
<td>4 30 34</td>
</tr>
<tr>
<td>chkn</td>
<td>918</td>
<td>0 806</td>
<td>0 112 112</td>
</tr>
<tr>
<td>cht</td>
<td>669</td>
<td>39 604</td>
<td>0 26 26</td>
</tr>
<tr>
<td>clpi</td>
<td>1108</td>
<td>27 964</td>
<td>3 114 117</td>
</tr>
<tr>
<td>clpl</td>
<td>38</td>
<td>0 14</td>
<td>0 24 24</td>
</tr>
<tr>
<td>cm138a</td>
<td>74</td>
<td>0 68</td>
<td>6 6</td>
</tr>
<tr>
<td>cm150a</td>
<td>245</td>
<td>23 222</td>
<td>0 0 0</td>
</tr>
<tr>
<td>cm152a</td>
<td>56</td>
<td>0 56</td>
<td>0 0 0</td>
</tr>
<tr>
<td>cm162a</td>
<td>166</td>
<td>6 113</td>
<td>0 47 47</td>
</tr>
<tr>
<td>cm163a</td>
<td>159</td>
<td>4 115</td>
<td>0 40 40</td>
</tr>
<tr>
<td>cm42a</td>
<td>76</td>
<td>0 68</td>
<td>2 6 6</td>
</tr>
<tr>
<td>cm82a</td>
<td>60</td>
<td>0 17</td>
<td>0 43 43</td>
</tr>
<tr>
<td>cm85a</td>
<td>131</td>
<td>0 107</td>
<td>0 24 24</td>
</tr>
<tr>
<td>cm89</td>
<td>141</td>
<td>6 92</td>
<td>0 43 43</td>
</tr>
<tr>
<td>con1</td>
<td>51</td>
<td>0 43</td>
<td>0 8 8</td>
</tr>
<tr>
<td>count</td>
<td>379</td>
<td>0 265</td>
<td>4 110 114</td>
</tr>
<tr>
<td>cu</td>
<td>164</td>
<td>6 100</td>
<td>27 31 58</td>
</tr>
<tr>
<td>dc1</td>
<td>120</td>
<td>0 85</td>
<td>7 28 35</td>
</tr>
<tr>
<td>dc2</td>
<td>255</td>
<td>0 201</td>
<td>3 51 54</td>
</tr>
<tr>
<td>decod</td>
<td>130</td>
<td>0 122</td>
<td>8 0 8</td>
</tr>
<tr>
<td>dist</td>
<td>796</td>
<td>0 712</td>
<td>4 80 84</td>
</tr>
<tr>
<td>duke2</td>
<td>1303</td>
<td>1 609</td>
<td>132 561 695</td>
</tr>
<tr>
<td>ex5</td>
<td>940</td>
<td>0 697</td>
<td>29 214 243</td>
</tr>
<tr>
<td>ex7</td>
<td>297</td>
<td>0 229</td>
<td>0 68 68</td>
</tr>
<tr>
<td>f51m</td>
<td>459</td>
<td>0 402</td>
<td>0 57 57</td>
</tr>
<tr>
<td>frg1</td>
<td>1049</td>
<td>0 1041</td>
<td>0 8 8</td>
</tr>
<tr>
<td>gary</td>
<td>1059</td>
<td>0 850</td>
<td>17 192 209</td>
</tr>
<tr>
<td>i1</td>
<td>129</td>
<td>1 89</td>
<td>2 37 39</td>
</tr>
<tr>
<td>ibm</td>
<td>492</td>
<td>0 354</td>
<td>0 138 138</td>
</tr>
<tr>
<td>in0</td>
<td>1059</td>
<td>0 850</td>
<td>17 192 209</td>
</tr>
<tr>
<td>in2</td>
<td>1002</td>
<td>0 757</td>
<td>56 189 245</td>
</tr>
<tr>
<td>in4</td>
<td>1013</td>
<td>0 696</td>
<td>22 295 317</td>
</tr>
<tr>
<td>in5</td>
<td>802</td>
<td>0 549</td>
<td>16 237 253</td>
</tr>
<tr>
<td>in6</td>
<td>767</td>
<td>0 548</td>
<td>26 193 219</td>
</tr>
<tr>
<td>in7</td>
<td>311</td>
<td>0 167</td>
<td>12 122 24</td>
</tr>
<tr>
<td>jbp</td>
<td>1132</td>
<td>0 833</td>
<td>55 244 299</td>
</tr>
<tr>
<td>lal</td>
<td>414</td>
<td>0 295</td>
<td>11 108 119</td>
</tr>
<tr>
<td>ldd</td>
<td>278</td>
<td>9 164</td>
<td>45 60 105</td>
</tr>
<tr>
<td>luc</td>
<td>621</td>
<td>0 430</td>
<td>55 136 191</td>
</tr>
<tr>
<td>m1</td>
<td>195</td>
<td>0 144</td>
<td>8 43 51</td>
</tr>
<tr>
<td>m2</td>
<td>543</td>
<td>0 442</td>
<td>7 94 101</td>
</tr>
<tr>
<td>m3</td>
<td>630</td>
<td>0 534</td>
<td>9 87 96</td>
</tr>
<tr>
<td>m4</td>
<td>973</td>
<td>0 844</td>
<td>8 121 129</td>
</tr>
<tr>
<td>majority</td>
<td>39</td>
<td>0 39</td>
<td>0 0 0</td>
</tr>
<tr>
<td>max46</td>
<td>380</td>
<td>0 380</td>
<td>0 0 0</td>
</tr>
<tr>
<td>max512</td>
<td>891</td>
<td>0 792</td>
<td>0 99 99</td>
</tr>
<tr>
<td>misex1</td>
<td>161</td>
<td>0 105</td>
<td>13 43 56</td>
</tr>
<tr>
<td>misex2</td>
<td>294</td>
<td>0 237</td>
<td>0 57 57</td>
</tr>
<tr>
<td>mlp4</td>
<td>694</td>
<td>0 590</td>
<td>92 122 104</td>
</tr>
</tbody>
</table>

Table 1: Detailed fault counts in two implementations
6.3. Points of vulnerability

The values $NFS$, $NST$, which give the number of points of vulnerability, are correlated very tightly between the two implementations. Figure 12 compares the numbers of Not Self-Testing faults, and Figure 14 compares the final vulnerability indicator, the Not Fault Secure faults. Tables 2 and 3 together

with Figures 11 to 14 indicate that while the LUT technology has bigger numbers of faults, most of them are in the B category, and that the number of faults in the C and D categories move in opposite directions, so that the number of vulnerable points—the NFS faults—remains almost constant.

It follows that the dependability, or, more precisely, the ability to become dependable using the MDS architecture, does not depend on architecture and fault model. Rather, it is a property of the circuit itself.

7. Conclusions

A method for proving arbitrary predicates quantified over an input vector of a combinational circuit has been presented. The method combines elements from SAT ATPG and SAT based property checking. The Modified Duplex System architecture, which requires classification into four classes, has been selected for demonstration of the method.

A set of benchmark circuits was constructed using the MDS redundancy architecture. The circuits were implemented both in gates and LUTs. Their self-checking characteristics were evaluated by the described method under the stuck-at and single event upset fault models, respectively. The characteristics were found to be correlated, which suggests that the ability to become dependable under the MDS scheme is an intrinsic property of the circuit itself.

Acknowledgement

This research has been supported by grant no. SGS12/094/OHK3/1T/18 from Ministry of Education, Youth and Physical Culture, Czech Republic.
Figure 14: NFS faults; $x = y$ provided for reference

References


URL http://dx.doi.org/10.1109/ISED.2011.70


