Evolving Redundant Structures for Reliable Circuits — Lessons Learned

Asbjoern Djupdal and Pauline C. Haddow
CRAB Lab (http://crab.idi.ntnu.no)
Department of Computer and Information Science
Norwegian University of Science and Technology
{djupdal,pauline}@idi.ntnu.no

Abstract

Fault Tolerance is an increasing challenge for integrated circuits due to semiconductor technology scaling. This paper looks at how artificial evolution may be tuned to the creation of novel redundancy structures which may be applied to meet this challenge. However, as these structures are unknown it is a challenge in itself to tune evolution to create them. As such, no solution has yet been found. This paper provides a discussion about the issues addressed and experiments conducted and thus provides an overview of the lessons learned in this work.

1 Introduction

As the semiconductor feature size decreases and the number of transistors on a single chip increases, one of the growing challenges facing the electronic design community is faulty behaviour [11]. This challenge may be met by either improved detection and repair techniques, by improved fault tolerance methods or a combination of the two.

The semiconductor fault challenge may be, in general, a long term challenge but is here today for large ICs, like Field Programmable Gate Arrays (FPGA). The mass production of FPGAs enables FPGAs to be produced in the newest technologies. Xilinx Virtex 5 [18] is an example of a new FPGA series from Virtex produced in 65nm technology with up to 330,000 logic cells.

If faults are expected to occur in a digital circuit, fault tolerance — the ability to function correctly in the presence of faults, may be achieved by incorporating redundancy (additional resources) in some form. These additional resources may be in the form of additional hardware, in which case it is called *hardware redundancy* [14] which is the focus of this paper.

One well known hardware fault tolerance method is Triple Modular Redundancy (TMR) [14]. This method involves tripling logic and using a voter to choose the correct solution. Although TMR is a very successful redundancy technique, its accepted weaknesses are the tripling of area and the susceptibility of the voter to faults. Also, if such a technique were to be applied to all the logic on an FPGA, around 2/3rds of the available logic would be applied to redundancy. This would drastically reduce the amount of primary logic on the device.

When introducing redundancy in a circuit, there is often a trade-off between area and fault tolerance. TMR may be said to trade area for higher fault tolerance. Much research is currently looking at more area efficient ways of achieving fault tolerance and section 2 provides a short summary of some of this work.

The goal for the work behind this paper is to find new ways of introducing redundancy in a circuit. The ultimate goal for this work is fault tolerance in FPGAs. While the work presented in this paper is not specifically targeting FPGAs, the goal is to gain knowledge on novel redundancy techniques that may later be adapted for use in an FPGA context. To find new redundancy techniques it is important to free oneself from the constraints brought upon us by thinking in the way of traditional redundancy. The whole way one thinks about designing at either the circuit design or technology architecture level is influenced by the way that one is taught electronics, designed electronics and the tools used in the design process. One way of freeing oneself from these human and design automated constraints is to search for ideas using some sort of heuristic search process. One such process is that of evolutionary algorithms (EA) [4].

The application of EA to the design of hardware is termed evolvable hardware [10]. The goal being, either to explore for unique solutions or to optimise existing solutions. However, in both cases, the goal is usually to obtain a given behaviour e.g. a binary adder [9]. Further, evolution may be applied when seeking some sort of structure, like evolving the french flag [16]. In both these cases the goal may be explicitly defined and given to the EA for comparison between the evolving solutions and the sought solutions. In the former case it is the functionality that needs to be explicitly defined whereas in the latter case the structure need be explicitly defined.

When evolving redundant circuits for the purpose of finding novel redundancy techniques, one is looking for redundant structures. However, these structures are unknown, unlike the case of the earlier mentioned french flag problem [16]. It is not possible to explicitly describe the structure that one is seeking, only the functionality of the sought circuit — perhaps in terms of the truth table.

In the earlier work of Hartmann and Haddow [7], fault tolerant circuits were evolved. While achieving high fitness on a reliability based fitness function, they did not focus on creating 100% functional circuits where reliability is achieved through redundancy. In this work, the goal is to push evolution to retain 100% functionality and to find ways to introducing redundancy for fault tolerance in the circuit. To the surprise of the authors themselves this problem is much more challenging than it might first appear. As such, the paper presents some of the approaches that have been applied to address this challenge and discusses these and further possibilities.

Section 2 gives a summary of the state of the art in area efficient redundancy techniques for FPGAs. Section 3 presents some important issues that must be addressed when evolving redundant circuits. Experimental setup, results and discussion is given in section 4 and the paper concludes in section 5.

2 Redundancy in FPGAs

To achieve area efficient defect tolerance, the typical approach is to exploit structural regularity [12]. The FPGA has a regular structure, which has inspired several techniques for defect tolerance in FPGAs. Techniques, especially in the context of enhancing yield, are reviewed in detail in [3]. Selected techniques may be classified under Node Redundancy, Configuration, Precompiled Configuration and Local Redundancy Techniques.

2.1 Node Redundancy

The node redundancy class of techniques contains the most widely studied techniques for redundancy in FPGAs and have been used with success to enhance yield in commercial Altera FPGAs [1]. The idea is to reserve spare nodes (logical blocks) in the FPGA architecture and enable spare nodes to take over for defective ones using on-chip resources. An early example [8] provided redundancy in the form of a redundant row. In the case of a defect, found during factory tests, the defective row is disconnected, vertical wiring is set up to bypass the disconnected row and all lower rows are shifted one row down. This reconfiguration is performed once, and may therefore be completed at the factory with antifuses or similar write-once technology.

The node redundancy technique has since been generalised to applying individual spare nodes instead of entire rows. A single defect then results in using one of the spare nodes [6], instead of discarding a whole row of nodes.

A difficulty with using such a method is to trade off hardware simplicity for good defect coverage. Changing the physical location of functionality from faulty functional nodes requires rerouting and flexible rerouting is expensive in terms of both area and delay and is difficult to achieve on-chip. Lack of flexibility either means that fewer defects are tolerated or that more redundancy is wasted on each of them.

2.2 Configuration

The FPGA, or a system external to it, may change its configuration so that defective portions of the chip are left unused. The concept here is to use spare resources that are naturally present in an FPGA design as no FPGA design uses all of the FPGAs resources. Doing a new place-and-route is very computationally expensive and uses much resources and is, therefore, often completed off-chip using standard synthesis tools (the Teramac project [2]). However, the Cell Matrix [15] provides an example of an on-chip solution incorporating a complex cell solution.

2.3 Precompiled Configuration

A further alternative to node redundancy is to share some of the work of rerouting with the synthesis tools. The bitstream may contain several different configurations, each assuming a defect in a different position. At configuration time, the chip may select those configurations that fits the current defect map [13].

2.4 Local Redundancy

The redundancy methods presented may be said to work at the system level. Local redundancy, on the other hand, introduces redundancy that effects only the area local to it i.e. adding extra local routing between two switch blocks. If a defective wire is found, the redundant one takes over its functionality [6, 19].

3 Issues on Evolving Redundant Structures

As stated, the goal of this work is to evolve redundant structures. There are, however, a number of issues that must be addressed and this section gives an introduction to these issues.

3.1 Behaviour (Function)

In this paper, the goal is not to evolve a multiplier, a flip flop or some other specified functionality. Instead our goal is to create redundant structures that enhances fault tolerance. However, we cannot explicitly define these structures but instead implicitly define them through a function that performs well in the presence of faults. As such, we need to define some kind of function and expose the function to faults. However, if the function is a challenge for evolution, evolution will use much time trying to achieve the function. This is, of course, undesirable. Instead, the function should be relatively easily evolvable so that evolution time is focused on the problem in hand, achieving redundant structures.

The size of the minimum representation of a given function is an important criteria when choosing a function. If there are very few gates, any form of redundancy will provide a substantial overhead, so a large circuit would probably be better suited for redundancy structures. However, this challenge has to be traded off against the challenge of evolving large circuits and wasting much evolution time on achieving the function itself rather than fault tolerance.

3.2 Fault Models

Two fault models are considered in this work: *the gate reliability model* and the *single fault model*. In the gate reliability model, each gate has a certain probability of failing. A *fault scenario* is one possible configuration of faulty gates for a given circuit. If a fault scenario for the gate reliability model is to be created, each gate in the circuit is tested against a random number generator and selected to be faulty or not based on a chosen gate reliability. This is a reasonable model of reality as the probability of having failing gates in a circuit is directly proportional to the number of gates in the circuit.

In the single fault model, a circuit can have exactly one fault at any time. If a fault scenario for the single fault model is to be created, one of the gates are selected to fail. Further, there are two cases of the single fault model. Either every possible gate failure is tested or a subset of possible gate failures may be tested to assess the reliability of the circuit in hand. The former is, of course, a more thorough and accurate test but uses significant resources. The latter is introduced with a view to reducing evaluation time.

3.3 Measuring Functionality and Reliability

The functionality of a circuit is found by trying all possible input values and recording the respective output values of the circuit. If all recorded output values correspond exactly to the desired truthtable for the function, the circuit is working perfectly, otherwise 100% functionality is not achieved. Traditionally, the result of such a test for functionality is either “not working” (0) or “working” (1), referred to as f_{bool} herein.

When using artificial evolution to create circuits, a measure of functionality is usually included in the fitness function. Since f_{bool} provides little information as to the quality of the solutions that are not 100% functional, evolution is unable to distinguish between two solutions that do not reach 100% functionality. Evolution needs to separate a circuit that is almost working, from a circuit that is far from working, even though both of these circuits will have $f_{bool} = 0$ (“not working”). One way of achieving more information is to measure the *hamming distance* between the circuit output and the desired output, i.e. the number of bits that are different between these two solutions. In this paper, the hamming distance is normalised to the interval $[0, 1]$ where 1 is 100% working. This measure of functionality is called f_{ham}. If a circuit is working, both f_{bool} and f_{ham} will be 1.

A reliability metric measures how well a circuit functions in the presence of faults. *The traditional reliability metric R_{trad} is the average of all f_{bool} results after having tested a number of randomly selected fault scenarios.* A second reliability measure can be formulated based on f_{ham}. *The reliability metric R_{ehw} is the average of all f_{ham} results after having tested a number of randomly selected fault scenarios.*

Table 1. Naming convention for reliability metrics together with fault models applied

Name	Meaning
R_{trad_single}	R_{trad} using single fault model
R_{trad_gate}	R_{trad} using gate reliability
R_{ehw_single}	R_{ehw} using single fault model
R_{ehw_gate}	R_{ehw} using gate reliability

The possible fault scenarios depend on the fault model chosen. The reliability of the circuit will depend on both the reliability metric and the fault model applied. To aid readability of the discussions and experiment the naming convention in table 1 is applied in this paper.

It is important to note that these reliability measures may be applied to provide a measure as to how well the circuit, when treated as a black box, performs in the presence of faults. It says nothing about the redundancy structures themselves which is, of course, the goal of this work. So how can one measure the presence of redundancy in a circuit? Automatic Test Pattern Generation (ATPG) tools may be used to identify redundancy as gates that can not be tested by any test vector and therefore represent redundant gates. However, this test detects redundancy whether useful (contributing to fault tolerance) or not. The question of detecting useful redundancy structures is in fact quite a complex one as highlighted herein.

4 Discussion, Experiments and Results

The purpose of this section is not only to present the experimental work in this paper, but also to present the process of analysing the problem itself and the intermediate results which led to the lessons learned.

4.1 Experimental Setup

All experiments are conducted on simulations of circuits in a digital feed forward circuit simulator. Only Boolean logic is allowed and the following gates are available: AND, OR, NAND, NOR, NOT. A faulty gate is simulated by inverting the output of the gate. The EA is Cartesian Genetic Programming [17] with the following parameters:

- Maximum number of gates: 200
- Population size: 20
- Tournament selection with elitism ($g = 3, p = 0.7$)
- Crossover rate: 0.2
- Mutation rate: 0.05 (mutation applied at the gate level)

4.2 Single Fault Experiments

In earlier work by Hartmann and Haddow [7], experiments were performed using the gate reliability model and circuits were evolved with a fitness function based on R_{ehw}. These experiments provided evidence that evolution traded off functionality for improving fitness. The number of gates were reduced to minimise the probability of having failing gates in the circuit, and instead of creating circuits that were 100% correct, simpler functions were created giving correct output for most of the possible input values.

Why did earlier experiments not lead to redundant structures, even though they were evolved with a reliability metric as fitness?

It seems that evolution chose the simplest way of attacking the problem — avoiding it by shrinking the circuit. When using the gate reliability model, the probability of having a failing gate is reduced when the number of gates is reduced. There is, therefore, an implicit size factor in the fitness evaluation that encourages small circuits, which is thus inhibiting larger circuits with redundancy.

These earlier experiments focused on making circuits that score high on the chosen reliability metric for given functions — multipliers and adders. Further work [5], also looked at the reliability metrics themselves and how they compare and may be applied in the context of traditional and evolved designs. As such, none of this work explicitly searched for redundancy structures and the goal herein is to find ways to either implicitly or explicitly specify a search leading to redundancy structures. Thus functionality and reliability in terms of a given metric are not in focus in this paper.

How could evolution be forced to create larger and more interesting redundant circuits?

One way of encouraging large circuits is to introduce a size factor in the fitness function that evens out the negative effect that a large circuit has on reliability. Introducing a size factor would require some sort of weighting between functionality and size. However, a size factor does not, in itself, encourage useful redundancy, just more gates.

Redundancy techniques typically introduce some kind of overhead, in terms of the number of gates, and this overhead is especially dominating for smaller circuits, such as those experimented with herein. Further, the number of gates in the circuit affects the probability that the circuit will have a gate that fails when applying the gate reliability model. However, applying the single fault model removes any bias towards smaller circuits.

The single fault model may be applied with either the R_{ehw} or the R_{trad} metric. As stated, R_{ehw} provides a measure of how a circuit degrades in the presence of faults. R_{ehw} may have a non-zero value even with faults present, whether or not the circuit has any redundancy or not. R_{trad}, on the other hand, is zero in the presence of a fault unless redundancy is present. As such R_{trad} may be said to be a better indicator of redundancy. R_{trad} does, however, not provide evolution with sufficient fine-grained information. A two part fitness function was thus created, as presented in equation (1). For the purpose of the experiments herein, k_1 was set to 0.3 and k_2 was set to 0.7.

$$f = k_1 \cdot f_{ham} + k_2 \cdot R_{trad\_single} \tag{1}$$

The first part containing f_{ham} takes care of building a functional circuit. Before f_{ham} is 1.0, R_{trad_single} will remain zero and thus does not contribute to the total fitness. When fitness = 0.3, a fully functional circuit is evolved and R_{trad_single} will be the part that evolution will have to increase in order to improve fitness.

As stated, a particular function is not the focus of the work herein. However, a function is needed for fitness evaluation. What function should be applied?

In this work, redundant structures are sought, requiring analysis of the evolved circuits after evolution. Earlier work [7] focused on $2 \cdot 2$ multipliers and adders. Having

Table 2. Results from R_{trad_single} experiments

#	Size	Fitness	R_{trad_single}	R_{trad_single} (opt)
0	72	0.862	0.792	0
1	111	0.936	0.901	0
2	90	0.906	0.856	0
3	40	0.803	0.700	0
4	132	0.941	0.909	0
5	99	0.907	0.859	0
6	68	0.854	0.779	0
7	96	0.926	0.885	0
8	46	0.798	0.696	0
9	65	0.891	0.831	0

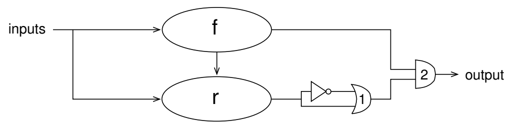

Figure 1. Redundancy structure exploiting unreachable gate

more than one output is from a functional point of view the same as having several circuits (even though logic may be shared between different outputs). Concentrating the evolutionary efforts on only one output thus seemed reasonable. In addition, analysis of a single output circuit is somewhat simpler than for a multiple output circuit.

The truthtable of a suitable function was constructed ("1001011101100110" with bit zero to the right), that is relatively easy to evolve, has four inputs and one output and with a non-redundant implementation of nine gates.

The results of ten evolution runs are given in table 2. Note that it would seem that evolution has been forced to use more gates and a reasonable fitness is achieved with R_{trad_single} lying between 0.7 and 0.9.

The results seemed promising until a manual inspection identified that what was being exploited by evolution was the concept of unreachable gates

An example of unreachable gates is shown in figure 1, where the ellipse marked *f* is a circuit performing our desired function. The OR gate marked *1* is unreachable. It will always have a constant one as output, no matter what the main circuit inputs are. The subcircuit having this unreachable gate as its output, marked *r*, will not affect the main circuit output in any way, so any faults in this area will be swallowed without creating a wrong output. Making a small functioning subcircuit *f* and making a large variant of such an unwanted subcircuit *r* will result in a high R_{trad_single}.

These *unreachable gate structures* are, however, not useful for our purpose. They contain no real redundancy. The

Table 3. Results from R_{trad_single} experiments where unreachable gate structures are avoided

#	Size	Fitness	R_{trad_single}	R_{trad_single} (opt)
0	106	0.933	0.896	0.896
1	135	0.937	0.904	0.904
2	135	0.937	0.904	0.904
3	61	0.883	0.820	0.820
4	61	0.883	0.820	0.820
5	108	0.921	0.880	0.880
6	109	0.922	0.881	0.881
7	108	0.921	0.880	0.880
8	101	0.923	0.881	0.881
9	101	0.923	0.881	0.881

last column in table 2, "R_{trad_single} (opt)", show the value of R_{trad_single} after the circuits have been optimised in such a way that the unreachable gate structures have been removed. The results show clearly that no other form of redundancy is present in these circuits.

The fitness value for circuits with unreachable gate structures is high. Such circuits are therefore the kind of circuits that are likely to be promoted to future generations. Even if circuits with good redundancy structures exist in early generations, they are probably discarded because the fitness function is again not explicit enough as to what a redundant structure is.

4.3 Single Fault, Excluding Unreachable Gates

Growing the before mentioned unreachable gate structures is probably the easy solution for the EA. Those structures should therefore be discouraged in some way so that other more useful redundancy structures may emerge.

How can evolution both be forced to increase the size of the circuits but not exploit unreachable gate structures?

Again, one might say that evolution has found a way to avoid the problem of gate faults but has not solved it. Removing the possibility of evolution creating unreachable gates would constrain evolution's freedom to explore for circuits. It was, therefore, deemed more appropriate to modify the fitness function such that unreachable gate structures do not contribute positively to the fitness value. This was achieved by detecting all gates that are part of a subcircuit with unreachable gates as outputs and when applying the single faults, faults were only applied to gates outwith these subcircuits.

A summary of the results of adapting the fault model to apply faults at only reachable gates may be found in table 3. As shown, it would seem that the problem of unreachable

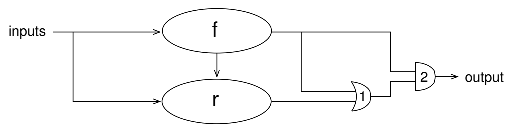

Figure 2. Reachable gates that do not contribute to the output

gates was solved and reasonable fitness was again reached.

However, evolution once again found a way to cheat.

When known unreachable gate structures were made unprofitable, another and similar kind of structure was invented by the EA. Instead of introducing unreachable gates at the exit point for a large random subcircuit, structures such as the example shown in figure 2 were created. Similar to the previous example, the ellipse marked *f* represents a circuit performing our desired function. Circuit *f* is connected to an AND gate (marked *2*). Evolution exploits the fact that the second input to this AND gate has “input don’t cares” whenever the first input is a logical 1. By introducing a structure, such as the one represented by gate *1*, evolution can once again grow a large circuit *r* that does not contribute to the output in any way, yet scores positively on fitness.

This is also an unwanted structure. It is, however, not as easy to detect automatically as the unreachable gate structures because there is an unlimited number of ways to construct similar solutions. It is, therefore, a challenge to exclude such structures from fitness evaluation.

4.4 Redundant Subcircuits

How can evolution be forced to put redundant structures within the circuit itself?

An alternative way of evolving redundant circuits is to split the target circuit into smaller subcircuits and evolve redundant versions of these subcircuits. This may provide evolution with a simpler problem to evolve and analysing these smaller circuits for redundancy might be simpler.

The problem of selecting subcircuits is not an easy one. What granularity to use? How to avoid illegal structures, like feedback loops? Since the goal of this work has nothing to do with partitioning, basic logic gates were selected as subcircuits so as to avoid the partitioning problem. Redundant versions of the basic logic gates were evolved and these together with the logic gates themselves were available to evolution to create a redundant circuit.

Again, evolution chose to find ways of introducing unreachable gates to the redundant logic gates in the same way as was introduced to the complete circuit (figure 2).

4.5 Larger Gate Reliability Experiments

When it was clear that using R_{trad_single} in the fitness function did not result in any useful redundancy structures, it was decided to try looking once again at the gate reliability model.

How could the complexity of the function sought be increased whilst avoiding the implicit size reduction in the fitness function and achieving a reasonable evaluation time despite applying the gate reliability model?

One of the challenges in using a gate reliability model is that there are a large number of possible fault scenarios. Using only a few fault scenarios drastically reduces the evaluation time at the expense of a very noisy fitness evaluation. A noisy fitness evaluation makes the task harder for evolution.

One possibility is to exploit the fact that the number of faulty gates in a fault scenario with the gate reliability model is binomially distributed. If X is a random variable for the number of faults in a fault scenario, x is the number of faults, n is the number of gates in the circuit and p is the fail rate for the gates ($1 -$ gate reliability), equation (2) may be used to find the probability of having a specific number of faulty gates in a fault scenario.

$$P[X = x] = b(x; n, p) = \binom{n}{x} p^x (1 - p)^{n-x} \qquad (2)$$

The circuit may be evaluated with the zero fault scenario and all the single fault scenarios and the results may be scaled by the probability for that number of faults ($x0$ and $x1$). However, the case of more than one fault would still be computationally expensive and, as such, it was chosen to express reliability excluding a component for more than one fault — see equations (3) and (4) for R_{trad} and R_{ehw} respectively.

$$R_{trad\_gate} = x0 \cdot f_{bool} + x1 \cdot R_{trad\_single} \qquad (3)$$

$$R_{ehw\_gate} = x0 \cdot f_{ham} + x1 \cdot R_{ehw\_single} \qquad (4)$$

The third output of the $3 \cdot 3$ multiplier was chosen because its non-redundant implementation needs 17 gates, as opposed to 9 in the earlier experiments. A fitness function was designed that ensures that a fully functioning circuit always scores higher than a circuit that does not function 100% correctly. This function is shown in equation (5). As shown, as long as the functionality is not 100% (f_{ham} is not 1.0), the R_{ehw_gate} part of the fitness function does not contribute to the fitness value.

$$f = k_1 \cdot f_{ham} + k_2 \cdot \begin{cases} 0 & \text{if } f_{ham} < 1.0 \\ R_{ehw\_gate} & \text{if } f_{ham} = 1.0 \end{cases} \qquad (5)$$

Table 4. Results from larger R_{ehw_gate} experiments

#	Size	Fitness	f_{ham}	R_{ehw_gate}	R_{trad_single}
0	23	0.941	1	0.921	0
1	22	0.946	1	0.928	0
2	23	0.944	1	0.928	0
3	23	0.946	1	0.928	0
4	23	0.936	1	0.916	0
5	26	0.940	1	0.924	0
6	24	0.940	1	0.923	0
7	23	0.928	1	0.907	0
8	25	0.943	1	0.928	0
9	24	0.943	1	0.928	0

The best fit individuals from nine evolutionary runs are shown in table 4. It is clear from the results that while evolution is able to make circuits that work 100% when no faults are applied (f_{ham} equals 1.0), they do not include any redundancy: R_{trad_single} is zero so no gates may fail without damaging the output. Instead of introducing redundancy, evolution tries to minimise the number of gates while still maintaining functionality. This is also obvious from the gate counts which are again significantly lower than those in tables 2 and 3, despite the larger circuit being evolved.

Is R_{ehw_gate} good enough at rewarding redundancy?

A benchmark that may be used for investigating how good the fitness function rewards redundancy, is to test the fitness function on a circuit and on a TMR-version of the same circuit. TMR is a known redundancy structure, and if a fitness function scores lower on the TMR version of the circuit it is an indication that the fitness function might also score lower for other redundant structures that might be considered interesting. A 17 gate example of the third output of the $3 \cdot 3$ multiplier was investigated in this context.

This benchmarking was performed both on R_{ehw_gate} and R_{trad_gate}, and the results are shown in table 5. The "quick" way of estimating the metrics is that described in equations (3) and (4) and applied during evolution. "MC" reflects a thorough Monte Carlo simulation of the circuit. It can be seen in the table that both R_{ehw_gate} and R_{trad_gate} (MC) are correctly favouring the TMR circuit, but that the "quick estimate" of R_{ehw_gate} is presenting the TMR poorer than the non-redundant version. This would indicate that the fitness function seems reasonable for R_{trad_gate} but not R_{ehw_gate}. This may be explained by the fact that with two or more faults appearing in a circuit, R_{trad_gate} is close to zero. However, R_{ehw_gate} will be significantly more than zero even for two or more faults. As such, the choice to exclude the "more than one fault" component of (4) is detrimental to R_{ehw_gate}.

When using the quick estimate, R_{trad_gate} seems better

Table 5. Benchmarking R_{ehw_gate} and R_{trad_gate} using $3 \cdot 3$ multiplier, output 3

		R_{ehw_gate}		R_{trad_gate}	
	Size	Quick	MC	Quick	MC
Plain	17	0.893	0.900	0.843	0.842
TMR	55	0.882	0.950	0.878	0.905

Table 6. Results from larger R_{trad_gate} experiments

#	Size	Fitness	f_{ham}	R_{trad_gate}	R_{trad_single}
0	18	0.884	1	0.843	0
1	18	0.884	1	0.829	0
2	21	0.867	1	0.809	0
3	22	0.861	1	0.805	0
4	20	0.873	1	0.817	0
5	21	0.867	1	0.809	0
6	18	0.884	1	0.832	0
7	21	0.867	1	0.812	0
8	21	0.867	1	0.812	0
9	21	0.867	1	0.811	0

suited for evolving redundant structures and this was tried experimentally. The fitness function applied is shown in equation (6) and the results are shown in table 6.

$$f = k_1 \cdot f_{ham} + k_2 \cdot R_{trad\_gate} \tag{6}$$

For these experiments, 100% functioning circuits are evolved but without any redundancy (shown by the zero values in the R_{trad_single} column). Instead, evolution tries its best to make the circuit as small as possible without removing functionality.

4.6 TMR Seeded Population

If it is hard to make evolution create redundant structures why not start at the other end and give evolution redundant structures and let it prune them?

The TMR experiments in this paper were conducted by seeding the starting population with one TMR-circuit. The rest of the population was random circuits. The experiments presented in the previous sections were rerun with TMR-seeded populations.

When using the R_{trad_single} based fitness function in equation (1) and avoiding unreachable gates, the original TMR structure of the seeding individual was kept. In addition, the EA introduced to the TMR circuit the kind of structure shown in figure 2, making it score highly on fitness. This experiment did however not result in anything new or useful.

Evolving for R_{ehw_gate} using the fitness function in equation (5) was also tried with TMR-seeded population and, interestingly, the TMR structure was not kept in this case. Instead, redundancy was removed. This is most likely caused by the fact that the quick estimator of R_{ehw_gate} is not favouring the TMR circuit over a non-redundant one.

When using R_{trad_gate} and the fitness function in equation (6) the TMR circuit is kept without any change at all. Evolution is unable to find any way of changing the TMR circuit that gets better fitness, and because of elitism, the same TMR circuit is kept as the best one.

5 Concluding Remarks and Future Work

Several different attempts at evolving redundancy structures were tried in this paper and the results illustrate the difficulty inherent in evolving redundant structures. When the single fault model was applied, new structures were created that scored high on fitness, but that provided no useful redundancy. When the gate reliability model was applied, evolution responded by making small circuits without any redundancy at all.

The challenge is to specify a fitness function that correctly scores high on circuits with useful redundancy structures and scores low on structures that are useless from a fault tolerance point of view, whilst still maintaining evolvability and encouraging the creation of such structures. One approach in this paper has been to actively avoid a known unwanted structure, only to discover that other forms of unwanted structures were invented instead. A better way would be to have a more general way of classifying redundancy as useful or not and only including useful redundant gates when calculating fitness. Further work will investigate better algorithms for detecting useful redundancy.

The functionality of the evolved circuits is not the focus herein, but the functionality may still affect how easily evolution can create redundancy structures. An interesting experiment would be to co-evolve the function itself together with the reliable circuits implementing this function.

In summary this work has shown that evolution “cheats” by avoiding the problem instead of attacking the problem aggressively.

References

[1] Altera. Apex redundancy. `http://www.altera.com/products/devices/apex/features/apx-redundancy.html`.

[2] W. B. Culbertson, R. Amerson, R. J. Carter, P. Kuekes, and G. Snider. Defect tolerance on the teramac custom computer. In *Proc. IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM)*, page 116, 1997.

[3] A. Djupdal and P. C. Haddow. Yield enhancing defect tolerance techniques for FPGAs. In *FPL 2007*, 2007. Submitted to FPL 2007.

[4] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[5] P. C. Haddow, M. Hartmann, and A. Djupdal. Addressing the metric challange: Evolved versus traditional fault tolerant circuits. In *Adaptive Hardware and Systems (AHS)*, 2007.

[6] F. Hanchek and S. Dutt. Methodologies for tolerating cell and interconnect faults in FPGAs. *IEEE Transactions on Computers*, 47(1):15–33, 1998.

[7] M. Hartmann and P. C. Haddow. Evolution of fault-tolerant and noise-robust digital designs. *IEE Proceedings - Computers and Digital Techniques*, 151(4):287–294, jul 2004.

[8] F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muraoga, A. Tanaka, and K. Kanzaki. Introducing redundancy in field programmable gate arrays. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 7.1.1–7.1.4, 1993.

[9] H. Hemmi, J. Mizoguchi, and K. Shimohara. Development and evolution of hardware behaviors. In *Artificial Life IV: Proc. 4th Int. Workshop Synthesis Simulation Living Syst.*, pages 371–376. MIT Press, 1994.

[10] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya. Evolving hardware with genetic learning: a first step towards building a darwin machine. In *Proc. 2nd int. conf. From animals to animats: simulation of adaptive behavior*, pages 417–424, 1993.

[11] ITRS. International technology roadmap for semiconductors. Technical report, ITRS, 2005.

[12] I. Koren and Z. Koren. Defect tolerance in VLSI circuits: Techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819–1837, sep 1998.

[13] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Low overhead fault-tolerant FPGA systems. *IEEE Trans. Very Large Scale Integr. Syst.*, 6(2):212–221, 1998.

[14] P. K. Lala. *Self-Checking and Fault Tolerant Digital Design*. Morgan Kaufmann Publishers, 2001.

[15] N. J. Macias and L. J. K. Durbeck. Adaptive methods for growing electronic circuits on an imperfect synthetic matrix. *Biosystems*, 73(3):173–204, 2004.

[16] J. F. Miller. Evolving a self-repairing, self-regulating, french flag organism. In *Genetic and Evolutionary Computation (GECCO)*, pages 129–139, 2004.

[17] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the evolutionary design of digital circuits part i. *Journal of Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[18] Xilinx. Xilinx virtex 5 overview. `http://www.xilinx.com/products/virtex5/index.htm`.

[19] A. J. Yu and G. G. F. Lemieux. Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement. In *Proc. Field Programmable Logic and Applications*, pages 255–252, 2005.