Evolving Efficient Redundancy by Exploiting the Analogue Nature of CMOS Transistors

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@idi.ntnu.no, pauline@idi.ntnu.no

Abstract

Fault tolerance is an increasing challenge for integrated circuits due to semiconductor technology scaling. Triple modular redundancy is often used to achieve fault tolerance in digital circuits, but this method is inefficient. By exploiting the analogue nature of CMOS transistors, more efficient redundancy techniques may be applied.

This paper looks at how artificial evolution may be guided towards the creation of redundancy structures at the CMOS transistor level. A preliminary experiment is performed that successfully evolves redundant stuck-open defect tolerant digital inverters.

Keywords: Evolvable hardware, Redundancy, Fault tolerance, FPGA

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 [1]. This challenge may be met by improved fault tolerance methods.

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* [2]. One form of hardware redundancy is *static* hardware redundancy. Static redundancy involves introducing extra components in a way that masks defects, thus without any need to detect and repair the defects. Triple Modular Redundancy (TMR) [3] is one well known static redundancy technique.

The semiconductor fault challenge may be, in general, a long term challenge but is here today for large ICs, like FPGAs. The mass production of FPGAs enables FPGAs to be produced in the newest technologies. Xilinx Virtex 5 [4] is an example of a new FPGA series from Virtex produced in 65nm technology with up to 330,000 logic cells. Just like other large lithographically produced chips, FPGAs suffer from production defects and would, from a fault tolerance point of view, benefit from redundancy.

Redundancy for improving fault tolerance may be included at the design level i.e. incorporating redundancy into the FPGA application. The generality of the FPGA architecture makes it possible for the application designer to use traditional Boolean fault tolerance techniques, e.g. TMR, in the circuit design. However, this increases the complexity of the circuit design. Further, such techniques are today only applied to chips that have already passed the yield test. If the problem of production defects increases to a point where known defective chips must be shipped, the requirement that the application designer must take care of tolerating these defects will be a hard one. The FPGA provides a bridge between chip production and the application designer. The inclusion of fault tolerance in the FPGA architecture itself would provide a functionally correct FPGA for the application designer, despite production defects. The application designer would thus be relieved from the complexity of designing for imperfect hardware.

One approach to achieving a fault tolerant FPGA architecture would be to change the high level FPGA architecture. One such high level architecture technique is to include a redundant row or column of logic blocks in the FPGA architecture. Such blocks are applied to take over if a defective row or column is detected [5]. Several such techniques are reviewed in [6].

Another way to make the FPGA fault tolerant is to attack the problem from the CMOS transistor level. Instead of looking at how high level components, such as logic blocks, may be structured to support a fault tolerance technique, the VLSI implementation of these high level blocks might themselves include fault tolerance. By attacking the fault problem at the transistor level, solutions not possible using Boolean techniques can now be considered. While the Boolean TMR technique is useful in many contexts, the technique is inefficient in that it requires a working voter in addition to three equal modules performing the desired function, of which two modules must work perfectly. If redundancy is introduced at the transistor level, more efficient redundancy techniques may be achieved.

Aunet and Hartmann [7] provide an example of efficient redundancy at transistor level illustrating that by having two equal circuits drive the same output, single stuck-open transistor defects are tolerated. The ganged CMOS minority gate was analysed in [8] and a version of the gate was presented where careful sizing of transistors together with redundant transistors results in tolerance to both stuck-closed and stuck-open defective transistors. The defect tolerance was achieved using three times the number of transistors and no voter was required. The Boolean alternative would be to use TMR, which would be less reliable and have larger area requirements due to the voter. In Schmid and Leblebici [9], modular redundancy, resembling TMR, is presented where the digital voter is substituted by an analogue averaging unit to improve reliability.

The focus of this paper is to find a way to create new static redundancy techniques for FPGAs by attacking the problem at the transistor level. 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 techniques. The way one thinks when designing circuits 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 [10]. The application of evolutionary algorithms to the design of hardware is termed evolvable hardware (EHW) [11].

Previous work [12] proved that it is possible to tune the evolutionary algorithm to create useful hardware redundancy. Not only was a TMR like redundancy structure created, but also some completely new redundancy structures. However, the work only considered Boolean logic and were constrained by the limitations in the Boolean world. Layzell and Thompson [13], on the other hand, have evolved circuits at the transistor level. Although not explicitly trying to evolve redundancy, a stuck-open fault tolerant digital inverter was evolved that contained redundant transistors in an efficient way similar to what was presented in [7]. Unlike Layzell and Thompson who did not try to force evolution to create efficient and useful redundancy, the goal of this paper is to find a way to explicitly evolve efficient redundancy structures at the transistor level. The purpose is not to evolve a specific fault tolerant circuit or to analyse some aspect of artificial evolution. Instead, the purpose is to seek examples of ways to use redundancy at the transistor level for achieving fault tolerance. Analysing these examples might then provide insight and inspiration that may lead to new techniques for creating more fault tolerant FPGAs.

The remainder of this paper is organised as follows: Section 2 discusses several aspects regarding evolution of redundancy structures using the analogue SPICE simulator. An experimental setup is presented in section 3 together with a preliminary experiment that demonstrates its usefulness. The paper concludes in section 4.

2 Evolving Circuits with Redundancy using SPICE

2.1 Fault Models

A *fault scenario* is one possible configuration of faulty transistors for a given circuit. A *fault model* dictates how the fault scenarios can be constructed and how probable the different fault scenarios are of occurring. Two fault models are considered in this work: *the transistor reliability model* and the *single fault model*.

In the transistor reliability model (the transistor equivalent of the gate reliability model in [12]), each transistor has a certain probability of failing and each transistor fails independently of each other. If a fault scenario for the transistor reliability model is to be created, each transistor in the circuit is tested against a random number generator and selected to be faulty or not, based on a chosen fault rate.

In the single fault model, a circuit can have exactly one fault at any time and any single fault scenario is equally probable. One and only one of the transistors are selected to fail for any given fault scenario.

A failing transistor can be modelled in several ways [14]. A well known model is the stuck-at model, where a failing transistor results in a wire being clamped to V_{ss} or V_{dd}. Another and more realistic model is where a failing transistor is either stuck-closed or stuck-open (permanently on or off). In this paper, stuck-open defective transistors are considered.

2.2 Measuring Functionality and Reliability

One functionality metric, f_{bool}, simply states whether the circuit functions correctly ($f_{bool} = 1$) or not ($f_{bool} = 0$). The output of a circuit is defined to be *true* if having a value larger than $\frac{V_{dd}}{2}$. If the output is less than $\frac{V_{dd}}{2}$, it is defined to be *false*. A circuit is said to be fully functioning if, for all possible input values, the output is correctly true or false according to a given truth table.

When evolving circuits, fitness may represent functionality in terms of how close the circuit's functionality is to the desired functionality. The main functionality metric for this paper, f_{rms}, is based on the Root-Mean-Square (RMS) error between the simulated output and the ideal output, for all n possible input values i.

$$f_{rms} = 1 - \sqrt{\frac{\sum_i^n (sim(i) - ideal(i))^2}{n}} \qquad (1)$$

A reliability metric indicates how well a circuit functions in the presence of faults. Reliability may be measured by testing the circuit against a number of randomly selected fault scenarios. The possible fault scenarios depend on the chosen fault model. The R_{trad} metric, which is used in this paper, is the percentage of these tests where the circuit is fully functioning ($f_{bool} = 1$). When R_{trad} is applied with the single fault model, it is named R_{trad_single}. R_{trad_single} may be calculated exactly by testing all possible single faults. When applied with the transistor reliability model, the metric is named R_{trad_trans}.

$$R_{trad\_trans} = x0 \cdot f_{bool} + x1 \cdot R_{trad\_single} \qquad (2)$$

R_{trad_trans} may be estimated using equation (2). $x0$ and $x1$ are the probabilities for having, respectively, zero and one defective transistor in a randomly chosen fault scenario. [15] provides an explanation and discussion of this estimator.

2.3 Evolving Redundancy

A *redundant transistor* in a circuit is a transistor that may fail without damaging the circuits outputs. To find if a transistor is redundant or not, a redundancy test may be performed where the transistor is temporarily made defective. If the circuit output is unaffected, the transistor is called redundant. Finding all redundant transistors in a circuit involves applying the redundancy test on all transistors one at a time.

Earlier work on evolving redundant structures [12, 15] has concentrated on evolving redundancy for Boolean logic. These experiments showed that it is far from straight forward to evolve useful redundancy. The work in [15] concluded that evolution chooses the easiest solution to the problem. When measuring fitness based on gate reliability in [15], the easiest solution was to minimise a 100% working non-redundant circuit. On the other hand, applying a single fault based fitness function resulted in large amounts of gates connected to the functioning circuit in a way that do not influence the output in any way. Such gates are redundant, but not in a useful way.

The single fault results in [15] were improved in [12] by introducing an algorithm that classifies a redundant gate as either "useful" or "fake". When evaluating fitness in this case, evolution started creating redundancy that had the potential to enhance fault tolerance. It was concluded that by carefully guiding evolution, evolution is able to produce circuits with useful redundancy. The experiments in [12] resulted

in both a voter structure resembling the TMR voter and some new redundancy structures.

2.4 SPICE Considerations

SPICE provides different ways of analysing circuits. For the work in this paper, two analyses are relevant: operating point analysis and transient analysis.

For performing an operating point analysis, the circuit inputs are modelled as voltage sources with voltage equal to either V_{dd} or V_{ss}. The operating point analysis provides the output voltage the circuit would have, assuming stable input values, and is performed for all possible input combinations.

When performing a transient analysis, the circuit inputs are modelled as Piece Wise Linear (PWL) voltage sources. PWL is used to analyse a specific input transition. As the circuit output may depend on what the previous input was, analysing different input transitions is important. When analysing an input transition, the PWL source starts at an initial input voltage and is then, during 1ns, swept to the input voltage that is to be analysed. This voltage is then kept stable for a specified amount of time. This specified amount of time is in effect the delay requirement for the circuit. SPICE analyses the circuits behaviour reporting the circuits output voltage at the end of the specified time interval. To find the functionality of a circuit using transient analysis, all possible transitions for all possible input combinations are performed. The least correct output is chosen and applied in equation (1). This is considerably slower than operating point analysis, but makes it possible to set a delay limit when evolving circuits because the output value is measured at a certain point in time.

3 Experiments and Results

3.1 Experimental Setup

Circuits are evolved using the SPICE simulator. A spice netlist is created each time a circuit is to be tested. This netlist is then written to a file to be read by the BSD licensed SPICE simulator *ngspice* [16] based on the original Berkeley SPICE3.

A representation resembling Cartesian Genetic Programming [17] is applied, with the modification that in a gene, both inputs and output of the component are explicitly defined, in addition to component type (nMOS or pMOS transistor) and transistor dimensions. A (1+4) evolutionary strategy is applied with mutation rate 0.1. Mutation is applied independently for each information block (inputs, output, type, sizes) inside each gene in the genome.

The BPTM 22nm CMOS transistor models are applied [18] with transistor sizes from 30nm to 1000nm. Feedback loops are not allowed, but several transistors may drive the same line.

The experiments are evolved in two phases. The reason for the first phase is to generate redundancy. Earlier work [15] has shown that the single fault model is best suited for generating redundancy. The first phase, the exploratory phase, starts evolving circuits from scratch using a single fault based fitness function:

$$f_1 = 0.3f_{rms} + 0.2R_{trad\_single} + 0.5f_{bool} \quad (3)$$

The first part in equation (3) is the functionality metric. This is to ensure correct functionality is achieved. The second part is the reliability metric. A R_{trad} reliability metric equals 0 until 100% functionality is achieved. To reward redundancy in a circuit before 100% functionality is reached, the functionality of the circuit is measured each time fitness is to be evaluated. The reliability metric is then calculated based on the current behaviour of the circuit, not the desired target behaviour. The last part of equation (3) is to ensure that once 100% functionality is reached, it stays there. Evolution is stopped after 12 hours run time.

As in [15], fake redundancy may be the result from evolving with a single fault fitness function. The algorithm applied in [12] for classifying redundancy as useful or fake can not be used in these experiments because it is too computationally expensive and not easily adapted to circuits where several components may drive the same line. Therefore, a second phase, allowed to run for 12 hours, applies a transistor reliability fitness function (equation (4)) so as to remove any fake redundant transistors. The best individual from phase one seeds the population.

$$f_2 = 0.3f_{rms} + 0.2R_{trad\_trans} + 0.5f_{bool} \quad (4)$$

The chosen target functionality for these experiments is a digital inverter. This inverter is to tolerate as many single stuck-open transistor faults as possible in an effective way. Stuck-open faults are chosen because the results are then easily compared to the techniques in [13] and [7]. It is known that these faults are possible to tolerate 100% using effective analogue "tricks", and for a preliminary experiment it is useful to know there is a potential for evolution to exploit. A stuck-open transistor is simulated by removing the transistor completely from the SPICE netlist.

Three different experiments are performed, one using operating point analysis and two using transient analysis with 99ns and 5ns delay requirements respectively. All three experiments involve running the same evolutionary setup ten times with different random seeds, to produce ten different individuals.

Table 1: Experiment summary

	Op	Trans99	Trans5
Avg. # generations	27270	8334	8506
Avg. f_{bool}	1	1	1
Avg. R_{trad_single}	1	1	1

Table 2: Best evolved inverters, operating point analysis, after 10 runs

#	Size 1	Size 2	Solution
0	5	4	Variant of fig. 2
1	8	4	Fig. 1
2	7	4	Fig. 1
3	7	4	Fig. 1
4	11	4	Fig. 1
5	9	4	Fig. 1
6	5	3	Fig. 2
7	7	4	Fig. 1
8	8	4	Fig. 1
9	5	3	Fig. 2

Table 3: Best evolved inverters, transient analysis, 99ns delay requirement, after 10 runs

#	Size1	Size2	Solution
0	3	3	Fig. 2
1	4	3	Fig. 2
2	6	4	Fig. 1
3	8	4	
4	5	5	Variant of fig. 1
5	6	5	Variant of fig. 1
6	9	5	
7	5	3	Fig. 2
8	6	4	Variant of fig. 2
9	6	5	Variant of fig. 1

Table 4: Best evolved inverters, transient analysis, 5ns delay requirement, after 10 runs

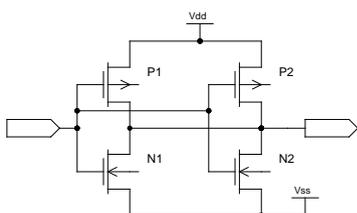| #	Size1	Size2	Solution
0	5	4	Fig. 1
1	6	4	Fig. 1
2	4	4	Fig. 1
3	8	4	Fig. 1
4	6	5	Variant of fig. 1
5	5	4	Fig. 1
6	5	4	Fig. 1
7	7	4	Fig. 1
8	6	4	Fig. 1
9	5	4	Fig. 1

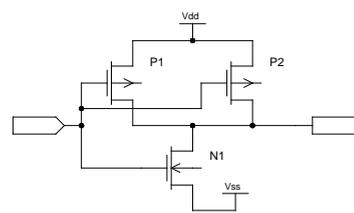**Figure 1:** Best evolved stuck-open tolerant inverter

Figure 2: Smallest evolved stuck-open tolerant inverter

3.2 Results and Discussion

Table 1 presents a summary of the results from the three experiments, showing average values for the best individuals in all ten evolutionary runs in each experiment. As can be seen, all evolutionary runs resulted in perfect working inverters ($f_{bool} = 1$) able to tolerate all single stuck-open faults ($R_{trad\_single} = 1$).

Table 2 shows a summary of the best individuals after ten evolutionary runs for the operating point analysis experiment. Tables 3 and 4 are similar but evolved using transient analysis for 99ns and 5ns respectively. "Size 1" is the number of transistors after phase 1 and "Size 2" is the number of transistors after having run the optimising phase 2. It is clear that in most cases, phase one introduces redundant transistors of which some are not helping improve R_{trad_trans} and, therefore, removed during phase 2.

Two main inverter variants are evolved, with some variations. The best evolved inverters resemble the one in figure 1. This solution showed up in all the three different experiments. This inverter is similar to a standard CMOS inverter, except that every transistor is duplicated such that two equal transistors drive the same output in parallel. This means that if one of the transistors is stuck-open but should have been conducting, the inverter output will still be driven correctly by the duplicate transistor.

The solution in figure 1 is the same as the one presented in [7]. This demonstrates that by rewarding redundancy in the way presented in this paper, it is possible to tune evolution to the creation of efficient redundancy structures. The solution in figure 1 is also the solution evolved in [13]. The redundant transistors in [13] were, however, introduced by evolution as a side effect of trying to improve fitness by creating a circuit with a slightly higher voltage swing. In contrast, the inverters in this paper contain redundancy because redundancy has been explicitly rewarded. While the result for stuck-open tolerant inverters is the same, it is expected that the experimental setup in this paper may also be applied to evolving redundant circuits tolerating other kinds of transistor faults, such as stuck-closed.

Figure 2 shows the other variant: an inverter with only three transistors. This solution showed up in both the operating point analysis experiment and the 99ns transient analysis experiment, but not in the 5ns transient analysis experiment. It is clear that the
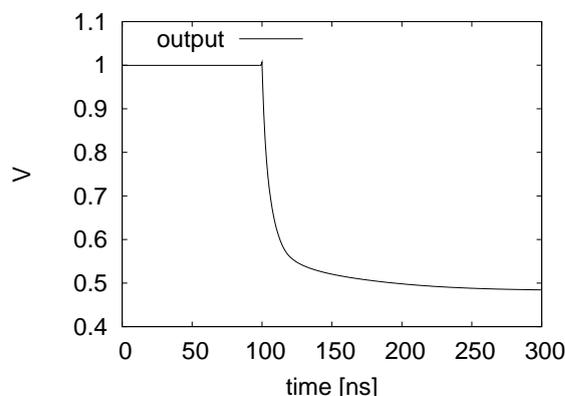

Figure 3: Transient analysis of the inverter in figure 2 when transistor N1 is stuck-open. The input changes from 0 to 1 at $t = 100ns$.

duplicated pMOS transistors provide reliability in the same way as the inverter in figure 1. However, can this inverter tolerate a stuck-open nMOS when only one nMOS is present? A transient analysis of this situation is given in 3. When the input changes from 0 to 1, the output is not driven by any active transistor. The output load, however, slowly discharges the output to a value slightly less than 0.5V. The circuit is therefore classified as working. This discharging takes longer than 5ns, which is why this solution is not created in the 5ns transient analysis experiment.

Tables 2–4 indicate which inverter variants are created in the different evolutionary runs. Although being simpler and faster, the operating point analysis experiment is just as successful at creating redundant inverters as the transient analysis experiments. When timing is of importance, transient analysis should be applied to ensure the circuit delay is small enough. When the goal is to produce lots of different examples of doing redundancy, operating point analysis might be better suited because of its faster run time.

4 Conclusion and Further Work

This paper has shown that a technique for evolving redundant structures, successfully applied earlier at the Boolean gate level [12], can also be applied with success, at the transistor level.

Experiments have been performed where digital inverters are evolved that are tolerant to stuck-open defective transistors. All evolved inverters

have achieved tolerance to stuck-open transistors by connecting several redundant transistors in parallel.

Further work is to include extensive experiments with the purpose of generating new and efficient redundancy structures.

5 References

[1] ITRS, "International Technology Roadmap for Semiconductors", Technical report, ITRS (2005).

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

[3] J. von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", in C. Shannon, editor, *Automata Studies*, pp 43–98 (1956).

[4] Xilinx, "Xilinx Virtex 5 Overview", http://www.xilinx.com/products/virtex5/index.htm (2007).

[5] F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muraoka, A. Tanaka and K. Kanzaki, "Introducing Redundancy in Field Programmable Gate Arrays", in *Proc. IEEE Custom Integrated Circuits Conference*, pp 7.1.1–7.1.4 (1993).

[6] A. Djupdal and P. C. Haddow, "Yield Enhancing Defect Tolerance Techniques for FPGAs", in *Int. MAPLD Conference* (2006), paper ID 203.

[7] S. Aunet and M. Hartmann, "Real-time Reconfigurable Linear Threshold Elements and Some Applications to Neural Hardware", in *Proc. Int. Conf. Evolvable Systems: From Biology to Hardware, (ICES)*, pp 365–376 (2003).

[8] A. Djupdal and P. Haddow, "Defect Tolerant Ganged CMOS Minority Gate", Submitted to NORCHIP 2007.

[9] A. Schmid and Y. Leblebici, "Robust and Fault-Tolerant Circuit Design for Nanometer-Scale Devices and Single-Electron Transistors", in *ISCAS*, pp 685–688 (2004).

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

[11] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware", in *Evolvable Systems: From Biology to Hardware (ICES 96)*, number 1259 in LNCS, pp 87–97, Springer (1996).

[12] A. Djupdal and P. C. Haddow, "Evolving and Analysing "Useful" Redundant Logic", in *ICES*, pp 256–267 (2007).

[13] P. J. Layzell and A. Thompson, "Understanding Inherent Qualities of Evolved Circuits: Evolutionary History as a Predictor of Fault Tolerance", in *Proc. International Conference on Evolvable Systems (ICES)*, pp 133–144 (2000).

[14] J. A. Abraham and W. K. Fuchs, "Fault and Error Models for VLSI", *Proc. IEEE*, 74(5), pp 639–654 (1986).

[15] A. Djupdal and P. C. Haddow, "Evolving Redundant Structures for Reliable Circuits – Lessons Learned", in *Adaptive Hardware and Systems*, pp 455–462 (2007).

[16] GEDA, "Ngspice homepage", http://ngspice.sourceforge.net/ (2007).

[17] J. F. Miller and P. Thomson, "Cartesian Genetic Programming", in *Genetic Programming, Proceedings of EuroGP'2000*, pp 121–132 (2000).

[18] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration", in *7th International Symposium on Quality Electronic Design (ISQED)*, pp 585–590 (2006).