Genet Program Evolvable Mach (2011) 12:281–303
DOI 10.1007/s10710-011-9129-2

The route to a defect tolerant LUT through artificial evolution

Asbjoern Djupdal · Pauline C. Haddow

Received: 7 September 2010 / Revised: 7 February 2011 / Published online: 2 March 2011

Abstract Evolutionary techniques may be applied to search for specific structures or functions, as specified in the fitness function. This paper addresses the challenge of finding an appropriate fitness function when searching for generic rather than specific structures which, when combined wiacteristic of defect tolerance on the circuit. Production defects for integrated circuits are expected to increase considerably. To avoid a corresponding drop in yield, improved defect tolerance solutions are needed. In the case of Field Programmable Gate Arrays (FPGAs), the predesigned gate array provides a bridge between production and the application designers. Thus, introduction of defect tolerant techniques to the FPGA itself could provide a defect free gate array to the application designer, despite production defects. The search for defect tolerance presented herein is directed at finding defect tolerant structures for an important building block of FPGAs: Look-Up Tables (LUTs). Two key approaches are presented: (1) applying evolved generic building blocks to a traditional LUT design and (2) evolving the LUT design directly. The results highlight the fact that evolved generic defect tolerant structures can contribute to highly reliable circuit designs at the expense of area usage. Further, they show that applying such a technique, rather than direct evolution, has benefits with respect to evolvability of larger circuits, again at the expense of area usage.

Keywords Defect tolerance · FPGA · Transistor level redundancy · Artificial evolution · Evolvable hardware · Fitness challenge

A. Djupdal (✉) · P. C. Haddow
CRAB Lab, IDI, NTNU, Trondheim, Norway
e-mail: asbjoern@djupdal.org

P. C. Haddow
e-mail: pauline@idi.ntnu.no

1 Introduction

The continued miniaturisation of features in CMOS integrated circuits (ICs) has resulted in larger, more complex and faster devices. The feature size in today's high-end chips is already at the nanoscale and is expected to scale further [19]. The lithographic process employed for the production of ICs can not be perfectly controlled, resulting in reduced yield due to production defects. The ITRS roadmap [19] predicts that this situation will worsen as CMOS is scaled down, resulting in a significant percentage of produced chips having defects. Predictions for future technologies are even more pessimistic where a significant portion of each produced chip is expected to be defective [26]. To handle the increasing defect rates and avoid yield levels resulting in prohibitively expensive chips, circuits should be designed to tolerate a certain amount of defective components. Such *defect tolerant* circuits could be achieved through the introduction of redundant components.

The Field Programmable Gate Array (FPGA) is a suitable target for redundancy techniques. FPGAs are today widely used, both for the original purpose as a prototyping device and as a component in end user products. High end FPGAs are produced with the most advanced production processes and are, therefore, among the first ICs that will encounter the expected increase in production defects. FPGAs can be seen as a bridge between production and the application designer. In a scenario where ICs must exhibit some kind of defect tolerance, an FPGA built with transparent redundancy techniques can provide the application designer with a functionally correct gate array, despite production defects. In addition, specialising the redundancy techniques towards the FPGA architecture can result in more efficient redundancy. To achieve area efficient defect tolerance, the typical approach is to exploit structural regularity [22]. The FPGA has a regular structure, which has inspired the search for effective defect tolerance techniques for FPGAs.

In traditional design, generic defect tolerant structures exist that may be applied to a circuit design to provide the design with fault/defect tolerance. However, finding new generic defect tolerant structures that are more defect tolerant, area efficient and/or power efficient than today's commonly used triple modular redundancy has proved challenging. To find new and innovative design solutions, the application of *Evolvable Hardware (EHW)* [17] techniques would seem appropriate. However, defect tolerance presents a challenging application for EHW. EHW, in general, is already challenged with respect to scaling up the technique to larger circuits—larger number of inputs. Adding faults and ensuring sufficient experiments for fault coverage, further exacerbates the fitness evaluation challenge and thus the viability of evolving such circuits.

The approach applied in this work is to apply evolution to the design of innovative generic defect tolerant structures that may be included in a traditional design, similar to the way triple modular redundancy is included today. However, such an application provides a further challenge for evolution. Traditionally, evolutionary techniques are applied to the evolution of given structures or functions, specified in the fitness function. However, in this case there is no given structure or function. Fitness needs to guide the emergence of a generic structure that when applied to a functioning circuit will provide for the property of increased reliability.

This paper is an extension of earlier papers by the authors. A potential new generic transistor level defect tolerance technique/structure, termed herein as the *Multiple Short-Open (MSO) Technique*, was presented by the authors in [10]. The MSO technique was the result of a manual analysis of circuits created with an *evolutionary Algorithm (EA)*. The process towards this technique, as well as the technique itself, is presented in Sect. 5 so as to highlight how EAs were applied in this process. Further, in a first step towards a fault tolerant FPGA, the structure is applied to an important building block of the FPGA, a Look-up table (LUT).

Two different strategies towards a defect tolerant LUT are followed in this paper. The first strategy is to apply the MSO technique to a traditionally designed LUT and the second is to evolve the defect tolerant LUT directly. Both strategies involve evolutionary experiments which share some of the same challenges regarding fitness functions and experimental setup. A large part of this paper is, therefore, devoted to describing and discussing these challenges.

Section 2 presents an overview of related work on traditional defect tolerance. Section 3 gives an introduction to artificial evolution and presents related work on evolved defect tolerance. Section 4 discusses a number of issues and practical considerations important for the evolution of transistor level circuits exhibiting redundancy. Section 5 presents the process that led to the MSO technique. Section 6 applies the MSO technique for constructing a defect tolerant LUT. Section 7 presents an evolutionary experiment where a defect tolerant LUT is evolved directly. A comparison of LUT implementations and a discussion is given in Sect. 8 and the paper concludes in Sect. 9.

2 Redundancy techniques for defect tolerant systems

One of the most popular and well known redundancy techniques is *Triple Modular Redundancy* (TMR) [24]. Three equal modules calculate the same function and a voter outputs the majority output. TMR is most often applied at the system level for critical systems with very low probability of failing. However, the reliability of a system is typically increased if redundancy is introduced at a more fine grained level. A system can be split up into smaller subsystems, each made redundant with TMR and cascaded to form the complete system. In a cascaded TMR system with small modules, a significant part of the system is devoted to voting. As such, the voter may need to be triplicated as well.

At the gate level, when the size of each submodule is only a few gates, TMR is unsuitable due to the voter becoming a dominant factor with respect to susceptibility to defects. A gate level alternative to TMR is *interwoven logic* [28]. Interwoven logic involves constructing the network of logic gates in a way such that it masks defects. Defect masking is achieved by quadrupling every gate in the system and connecting the gates in a specific way so as to avoid the need for a voter.

If very high reliability is required, redundancy can be introduced at the transistor level. Transistor level redundancy can be applied to build reliable components that form the basis for higher level redundancy techniques. One transistor level redundancy technique known as series-parallel transistor replication is shown in

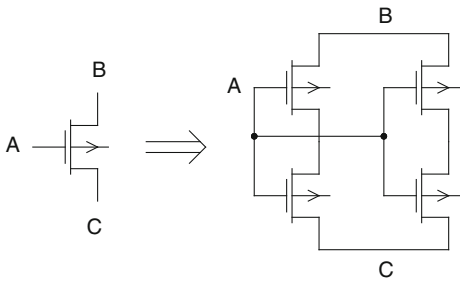


Fig. 1 Series and parallel replication of transistors

Fig. 1. Originally described by Moore and Shannon for relays [27], the technique provides redundant transistors in series for tolerating stuck-closed defects and redundant transistors in parallel to tolerate stuck-open defects. Combining these, as in Fig. 1, results in tolerance to both stuck-open and stuck-closed faults. TMR may also be applied at the transistor level. However, when considering only stuck-open and stuck-closed defective transistors at high defect rates, the series-parallel technique results in more reliable circuits than TMR [3]. Although the series-parallel technique tolerates all single stuck-open and stuck-closed defects, other possible defects, such as transistors with shorted gate and source, can still be catastrophic.

Bolchini et al. [2] present another example of transistor level redundancy targeting multiple output static CMOS circuits. Single stuck-closed defects are tolerated and a number of other defects are detected through the application of Berger codes.

One benefit of introducing redundancy at the transistor level is the possibility of exploiting non-digital properties of the technology [8]. One example is found in earlier work by the authors [7] where a minority gate was made tolerant to stuck-open and stuck-closed defects through a redundancy technique not possible at the digital gate level. Bolchini et al. [4] presents another example of transistor level redundancy not possible at the gate level, providing tolerance to stuck-closed defective transistors.

2.1 Defect tolerance for FPGAs

The most widely researched defect tolerance technique for FPGAs is variants on the redundant row technique, originally proposed by Hatori et al. [16]. One row of logic blocks is reserved as a spare row. If a defect is found in a row, the defective row is bypassed and the spare row is put into use. A variant of the redundant row technique is employed by Altera to enhance yield in some of their commercial FPGAs [1].

Most of the redundancy techniques for FPGAs, including the redundant row technique, may be said to work at the chip level. However, it is also possible to apply redundancy internal to the basic building blocks of the FPGA, such as the logic blocks and switch blocks. KleinOsowski and Lilja [21] explore both the application of error correcting codes and TMR to enhance the reliability of LUTs. Saha et al. [29] suggests introducing error correcting codes to the LUTs in the Cell

Matrix architecture. Doumar and Ito [11] add an extra wire to the switch block to provide a bypass for a faulty switch.

Defect tolerance techniques for FPGAs, especially in the context of enhancing yield, are reviewed in detail by the authors in [6].

3 Evolution of defect tolerant circuits

Several researchers have applied evolvable hardware (EHW) [17] in the search for fault or defect tolerant circuits. One approach is to evolve new solutions when the old one no longer functions, e.g. [12, 20, 32]. A defect that results in faulty behaviour triggers a mechanism that starts the evolution of a new circuit that can cope with the defect.

A further approach to achieve defect tolerant FPGAs is through static hardware redundancy. The goal of this work is static redundancy for LUTs and the static strategies for evolved fault and defect tolerance are, therefore, more relevant to this paper.

Thompson [30, 31] pioneered the field of evolved fault tolerance with an evolved state machine for a robot controller, capable of tolerating stuck-at faults in the 32 bit large RAM implementing the state machine. Thompson was able to evolve a static solution that was tolerant to any single stuck-at fault in the RAM. Canham and Tyrrell [5] evolved an oscillator for a Xilinx Virtex FPGA exhibiting redundancy to tolerate simulated stuck-at and bridging faults. Hartmann and Haddow [15] have evolved gate level circuits targeting tolerance to both noise and faults. Keymeulen et al. [20] have evolved fault tolerant transistor level circuits for a field programmable transistor array (FPTA), resulting in analog multipliers and digital XNOR gates tolerant to six predefined defects in the FPTA. Layzell and Thompson [23] present another example of evolved transistor level defect tolerance for digital gates. Although a byproduct of their fitness function, they evolved digital inverters where parallel replication of transistors provides tolerance to stuck-open defects.

The work of Hilder et al. [18] is, as in this paper, concerned with the yield issue, applies evolution to transistor sizing and applies spice simulations to evaluate designs. Rather than adding defect tolerant structures to an otherwise potentially unreliable circuit, the intention is to create reliable components for a *standard cell library (SCL)*. SCLs are libraries of components, where each component is a pre-designed circuit, and such components may be selected and combined to create a circuit. Using statistically enhanced SPICE models based on 3D-atomistic simulations, a multiobjective optimisation algorithm is applied to evolve the transistor dimensions of the circuit (component) so as to tolerate random fluctuations prevalent in future technology.

4 Issues on evolving transistor level redundancy

The approach taken for evolution of redundant circuits in this paper follows the technique outlined in [8]. When fitness is to be evaluated, the circuit is tested

repeatedly with different injected defects. Defect injection during fitness evaluation provides a means to calculate a reliability metric for the circuit. The quality of the evolved redundancy depends on the way defect injection is performed and how the reliability metric is included in the fitness function.

4.1 Measuring functionality

To measure if a circuit is functional according to a specification, test vectors are applied at the circuit's inputs and the response on the output is monitored with SPICE simulations. 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*. One functionality metric, f_{bool}, simply states whether the circuit has a correct Boolean response to all tested input vectors ($f_{bool} = 1$) or not ($f_{bool} = 0$).

f_{bool} is an important functionality metric when reliability is to be determined. However, f_{bool} might be too coarse grained when evolving towards a specific functionality. To avoid the evolutionary algorithm from becoming a random search, the fitness function must provide enough information for separating good and poorer individuals, even when no individual in the population has reached 100% functionality. Fitness should, therefore, include a functionality metric that represents functionality in terms of how close the circuit's output voltage is to the desired output voltage. 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 output measurements.

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

An illustration of the functionality test setup is given in Fig. 2. To make sure the evolved circuits are able to drive a representative load, the output of the circuit under test is connected to a chain of two inverters. Inverters are also driving the inputs to the circuit under test to avoid using perfect voltage sources as inputs. Perfect voltage sources would not be representative when an injected fault results in a short between input and either V_{dd} or V_{ss}.

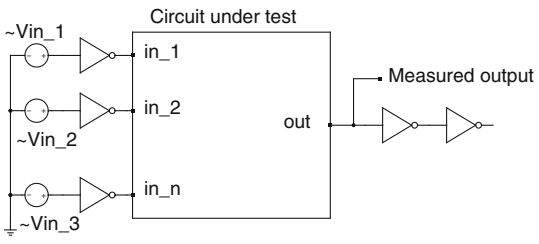


Fig. 2 Functionality test setup

4.2 Fault models

A *fault scenario* is one possible configuration of faulty transistors for a given circuit. The term *fault model* is applied in this paper as the specification of 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, 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.

4.3 Failing transistors

A transistor may fail in several ways. In this paper, several types of transistor defects are considered: Stuck-open transistors are permanently off and are modelled by connecting the gate to V_{dd} for pMOS transistors and V_{ss} for nMOS. Stuck-closed transistors are permanently on and are modelled by connecting the gate to V_{ss} for pMOS transistors and V_{dd} for nMOS. In addition, there may be a short between gate/drain or gate/source which both are modelled with a 1Ω resistor shorting the respective transistor terminals. It should be noted that a 0Ω resistor would have been preferable but not allowed for in the simulator. However, the 1Ω resistor could have been replaced with a smaller resistor.

4.4 Measuring reliability

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}. When applied with the transistor reliability model, the metric is named R_{trad_trans}. R_{trad_trans} can be said to be the probability of functioning 100%, given a certain transistor fail rate.

R_{trad_single} may be calculated exactly by testing all possible single faults. R_{trad_trans} may be estimated using a Monte Carlo simulation. R_{trad_trans} must, however, be estimated once for every fitness evaluation for every individual in the population during the entire evolutionary experiment. The result is that thousands of R_{trad_trans} estimations must be performed for every evolutionary run. A thorough Monte Carlo simulation is, therefore, too time consuming during evolution. One possibility is to exploit the fact that the number of defective transistors in a fault

scenario with the transistor 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 transistors in the circuit and p is the fail rate for the transistors, (2) may be applied to find the probability of having a specific number of defective transistors in a fault scenario.

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

To find the reliability of a circuit, the circuit is 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 defects ($x0$ and $x1$). This is shown in (3).

$$\begin{aligned} R_{trad\_trans} = {} & x0 \cdot f_{bool} + x1 \cdot R_{trad\_single} \\ & + (1 - x0 - x1) \cdot R_{MC>1} \end{aligned} \tag{3}$$

The reliability of the circuit when having more than one defect ($R_{MC>1}$) must still be found through Monte Carlo simulations. However, if $(1 - x0 - x1)$ is small, the number of Monte Carlo tests can be greatly reduced. If $(1 - x0 - x1)$ is close to zero, the $R_{MC>1}$ part of (3) may be ignored completely.

4.5 Fitness function

Earlier work on evolving transistor level redundancy [8] achieved best results when evolving the circuits in two phases. First generate redundancy using an R_{trad_single} based fitness function. As concluded in [9], an R_{trad_single} based fitness function is better suited for generating redundancy than an R_{trad_trans} based fitness function. Phase one typically generates very bloated circuits. The evolved circuit is, therefore, optimised in a second evolutionary phase using an R_{trad_trans} based fitness function. R_{trad_trans} is much less forgiving for transistors without any real purpose.

The following two fitness functions, f_1 and f_2, are applied in this paper for phase one (4) and phase two (5):

$$f_1 = k_1 f_{rms} + k_2 \hat{f}_{rms} + k_3 R_{trad\_single} + k_4 f_{bool} \tag{4}$$

$$f_2 = k_1 f_{rms} + k_2 \hat{f}_{rms} + k_3 R_{trad\_trans} + k_4 f_{bool} \tag{5}$$

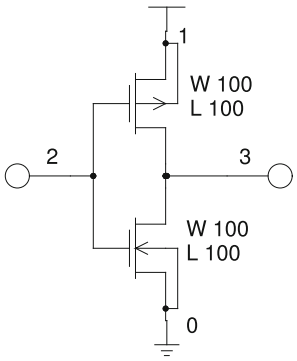
The first component, f_{rms}, is for a single test with no defective transistors and is included to guide evolution towards a functioning circuit with high output voltage swing. The second component, $\hat{f}_{rms}$, represents the average f_{rms} after having tested the circuit for all single faults. The second component is included to encourage high output voltage swing also when there are defective transistors. The third component is the reliability metric and the fourth component, f_{bool} is to make sure a working circuit is always rewarded more than a non-working circuit.

4.6 Evolutionary algorithm

Two different evolutionary algorithms are applied in this work. In the first experiments a $(1 + 4)$ evolutionary strategy is employed and in the later

```
<id>Inverter</id>
<inputs>1</inputs>
<outputs>1</outputs>
<netlist>
  <fu>
    <type>PMOS</type>
    <width>100</width>
    <length>100</length>
    <input0>2</input0>
    <input1>1</input1>
    <output>3</output>
  </fu>
  <fu>
    <type>NMOS</type>
    <width>100</width>
    <length>100</length>
    <input0>2</input0>
    <input1>0</input1>
    <output>3</output>
  </fu>
</netlist>
```

(a) XML



(b) Graphical

Fig. 3 Example of a genome

experiments a genetic algorithm with tournament selection. A *Cartesian Genetic Programming (CGP)* [25] like representation was applied in both cases as shown in XML in Fig. 3. An example genome is shown in Fig. 3. As is traditional for CGP, each node <fu> represents a gene and is described in terms of its input and output nets and a type. In addition, each node has a value for width and length.

There is no explicit definition of a net, a net is implicitly formed when a gene references a net by the same number as in another gene. The numbered ID is, therefore, uniquely identifying a net. Some nets have fixed IDs: V_{ss} has ID 0 and V_{dd} has ID 1. The next IDs represent the inputs to the circuit (ID 2 in the example). The output net of the last gene in the genome represents the circuit output (ID 3 in the example). All other IDs are generic nets. The transistor well is always connected to the source. In addition, transistor type (nMOS or pMOS) and characteristics (gate dimensions) are specified for each transistor. The genome can be directly translated to a SPICE netlist for simulation. The genome can specify any circuit topology, including feedback loops.

Mutation is applied independently for each information block (inputs, output, type, sizes) inside each gene in the genome. This means that a mutation that changes e.g. the output connection will not affect any other parts of the gene. Mutation is performed by simply exchanging the value with a random value within predefined limits.

For crossover, the genome can be seen as a string of genes. The ordering of this string is independent of the connections the transistors have with each other. Crossover is performed on gene boundaries, splitting each genome in two at a randomly selected spot, joining the resulting parts together.

There are several different experiments in this paper. The details of the evolutionary algorithm (selection, mutation rate, population size etc.) are specified when the different experiments are explained.

5 Multiple short-open technique

The series-parallel technique described in Sect. 2 is designed for tolerance to stuck-open and stuck-closed defective transistors. Other defect types can, however, still be catastrophic. One example is transistors where the gate is shorted to either source or drain. To tolerate such defects, a new redundancy technique was introduced in [10]. The technique provides tolerance to any short between two of the three transistor terminals and any open on any of the three transistor terminals. The new technique is herein termed the *Multiple Short-Open (MSO) technique* and is presented in Sect. 5.3. The process towards the technique is also presented as the process highlights one way of applying evolution for the creation of redundancy structures.

5.1 Experimental setup

The first step towards the new redundancy technique is to evolve a circuit with successful redundancy. To be able to measure redundancy, a functional circuit is required which can then be exposed to faults and the tolerance measured. Further, to keep the circuit complexity low (and thus the evolution time), the chosen target functionality was a digital inverter.

When the functionality of a circuit is to be tested, all possible input transitions are tested in turn by setting the Piece Wise Linear (PWL) input voltage sources in Fig. 2 to correspond to the input transition to be tested. A transient analysis of the test setup is then performed in the BSD licensed SPICE simulator *ngspice* [13].

The circuit output is measured after inputs have been stable for 50 ns. Circuit components allowed are nMOS and pMOS transistors expressed in the format illustrated in Fig. 3. The V1.0 Berkeley Predictive Technology Model (BPTM) 22 nm CMOS transistor models [33] are applied with allowed transistor sizes from 30 to 1,000 nm. Supply voltage $V_{dd} = 1V$.

A (1 + 4) evolutionary strategy is applied with mutation rate 0.1. Evolution may create the circuit from a maximum of 50 transistors and 55 nets for each circuit. A net is an internal wire in the circuit, including inputs and output. Fault scenarios are created, employing the following defect types: drain-source short (stuck-closed), gate-drain short and gate-source short. It should be noted that this is not an exhaustive list of defect types.

R_{trad_trans}, a component in the fitness function for evolution phase two (equation (5)), can only be found given a certain transistor reliability. The transistor reliability applied in this experiment is 0.99, a relatively low number chosen because of the small size of the circuits in this paper. Coefficients used for the fitness functions for both evolution phases are $k_1 = k_2 = k_3 = 0.2$ and $k_4 = 0.4$. The high value for k_4 is there to favour fully functioning circuits over non-functioning circuits.

5.2 Analysis of best evolved inverter

The best circuit found by evolution is shown in Fig. 4. The evolved inverter is fully functional and characteristics of the inverter are shown in Table 1. As seen by the R_{trad_single} metric, the inverter is tolerant to all possible single gate/drain, gate/source and source/drain shorts on any transistor present in the circuit. As such, the circuit in Fig. 4 is suited for further analysis.

The standard CMOS inverter is shown in Fig. 5 and consists of a pull-up pMOS transistor (M1) and a pull-down nMOS transistor (M2). The first step towards understanding the evolved circuit is to identify the corresponding pull-up and pull-down transistor networks. Transistors M1, M2, M6 and M7 in Fig. 4 represent the pull-up network, while transistors M12 and M15 represent the pull-down network.

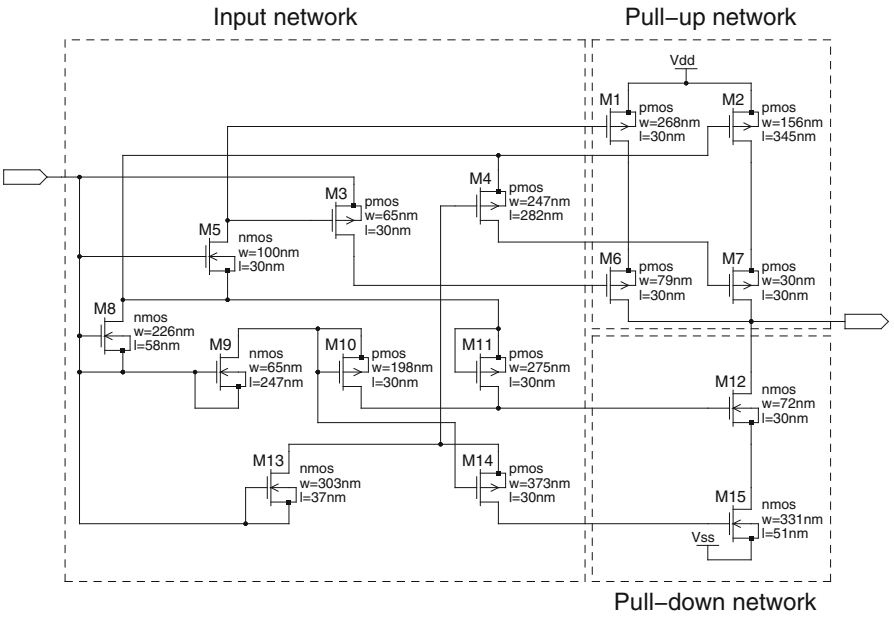


Fig. 4 Evolved defect tolerant inverter

Table 1 Characteristics of inverter in Fig. 4

Property	Value
Size	15 transistors
f_{rms}	0.998929
$\hat{f}_{rms}$	0.978471
R_{trad_single}	1.000000
$R_{trad\_trans} \vert R_t = 0.99$	0.965700

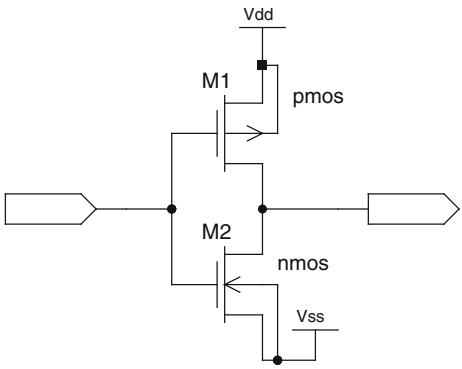


Fig. 5 Standard inverter

The pull-up and pull-down structures are interesting by themselves. First, they show that evolution has introduced redundant transistors in series (M1/M6, M2/M7, M12/M15) to tolerate drain/source shorts (stuck-closed transistors). As the drain/source short defect was one of the defects injected during evolution, redundant transistors in series was expected.

However, evolution also introduced redundant transistors in parallel in the pull-up network (the M1–M6 chain is parallel to M2–M7), a structure known to tolerate stuck-open defective transistors. Stuck-open was *not* one of the defects injected during evolution, so why did evolution introduce these parallel structures? When there is a short between gate and source on transistor M1 or M2, the transistor is effectively stuck-open, resulting in the need for a parallel chain of transistors. The same reasoning applies for the pull-down network. However, instead of introducing parallelism in the pull-down network, evolution has relied on the output slowly discharging to the correct value. Unfortunately, such a solution is suboptimal because the delay will increase and the output will just barely reach a value less than $\frac{V_{dd}}{2}$.

The next step is to understand the purpose of the transistors in the input network i.e the transistors that connect the inverter input with the transistor gates in the pull-up and pull-down networks. None of these transistors are connected to V_{dd} or V_{ss}, but are instead just passing on the inverter input. SPICE simulations showed that all nets in the input network are more or less degraded versions of the inverter input. It seems that evolution has tried to separate the inverter input from the pull-up and pull-down networks with a resistive circuit. To tolerate a short between, for example, the transistor M1 gate and source, the inverter input must be separated from the gate to avoid clamping the input to V_{dd} and thus resulting in the inverter output stuck-at-0. If the input is separated from the shorted gate with a resistor, the result is a slightly degraded input signal whilst retaining correct output.

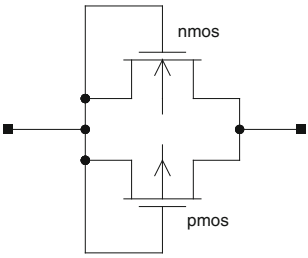
A resistor in a CMOS IC can be formed with an nMOS transistor with gate connected to V_{dd}. Evolution never introduced such resistors in the input network in Fig. 4 because those resistors are not themselves tolerant to gate/source and gate/drain shorts. Instead, evolution has created the input network without any connections to V_{dd} or V_{ss}, thus avoiding the problem of gate shorts in the input network.

5.3 A generalised defect tolerance technique

The analysis in Sect. 5.2 was then used as a basis to form the new redundancy technique: (1) To allow for stuck-open and stuck-closed transistors, redundant transistors should be introduced to the pull-up and pull-down networks both in series and parallel, as in Fig. 1, (2) To tolerate gate/source and gate/drain shorts, the transistor gates in the pull-up and pull-down networks must be isolated from the inverter input using a defect tolerant resistor. A defect tolerant resistor can be formed with a high-resistivity polysilicon meander structure but is here implemented with two transistors as shown in Fig. 6. Combining Fig. 1 and the defect tolerant resistor results in the MSO technique summarised in Fig. 7.

To demonstrate how the MSO technique can be used to create a defect tolerant inverter, Fig. 8 shows the result after having applied the substitution in Fig. 7 to the standard inverter in Fig. 5. The resistance of the resistors must be large enough to achieve isolation. For the circuit in Fig. 8, minimum sized transistor gates are suitable for most of the resistors, except for R7 and R8 that should be sized for larger resistance to reduce the impact of a gate short to V_{ss}.

Fig. 6 Resistor implementation



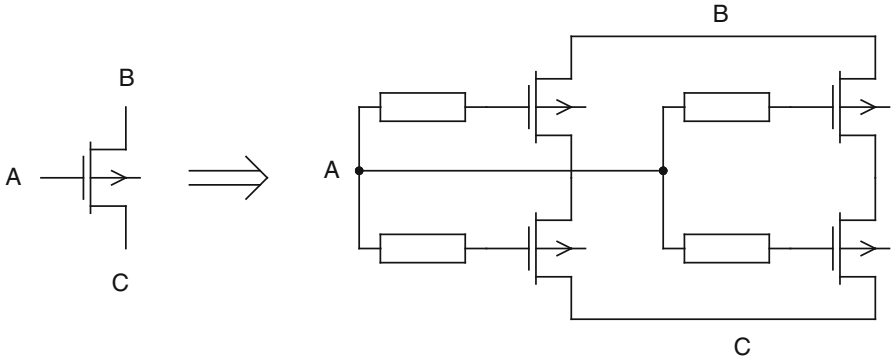


Fig. 7 The multiple short-open (MSO) technique, shown for pMOS. nMOS transistors are substituted in the same way. Resistors can be implemented as in Fig. 6

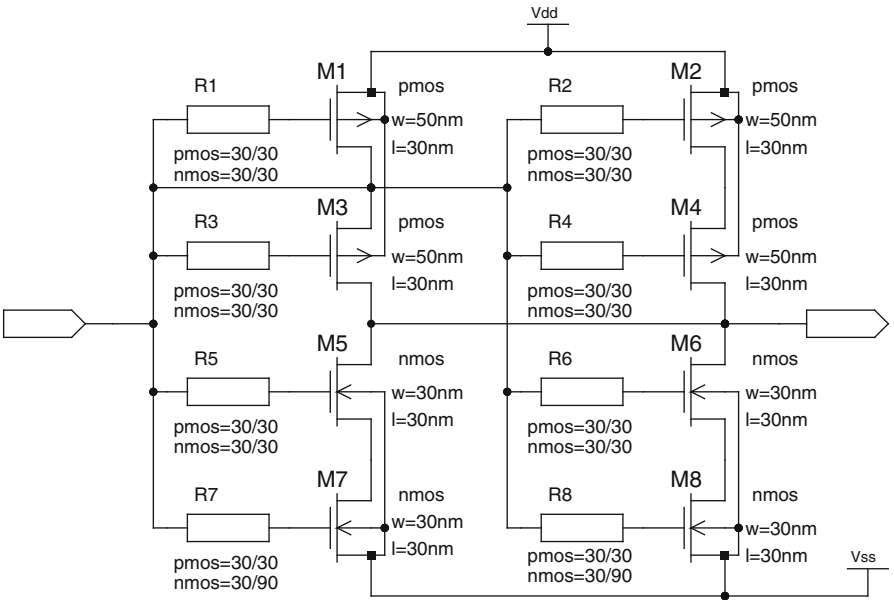


Fig. 8 Defect tolerant inverter. Resistor sizing given as W/L

6 Defect tolerant LUT based on the MSO technique

The MSO technique described in Sect. 5 can now be applied when implementing a LUT with traditional design techniques. To keep the size and complexity of the LUT at a manageable level for the evolutionary experiment in Sect. 7, the chosen LUT specification for this paper is one with only one address input, referred to as LUT1. A traditional implementation of LUT1 is shown in Fig. 9, consisting of two standard 6-transistor SRAM cells and a standard 8-transistor static CMOS multiplexer. Figure 9 also shows which inputs and outputs are required for LUT1. Asserted "W0" or "W1" results in the value on "D" (and its complement "D") is written to the respective SRAM cell. "A" chooses which SRAM value should be reflected on the output.

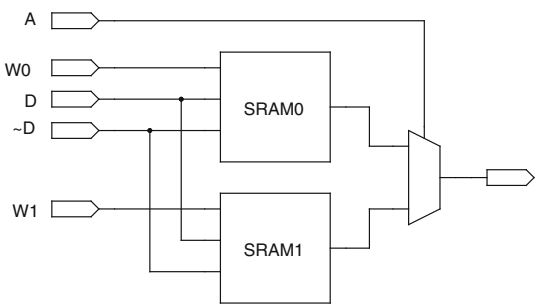


Fig. 9 Traditional implementation of LUT1

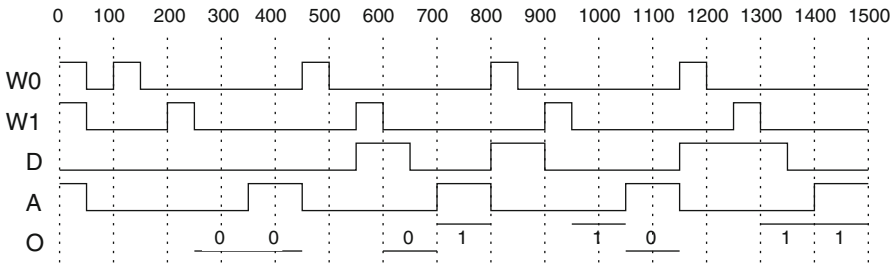


Fig. 10 Functionality test of LUT1

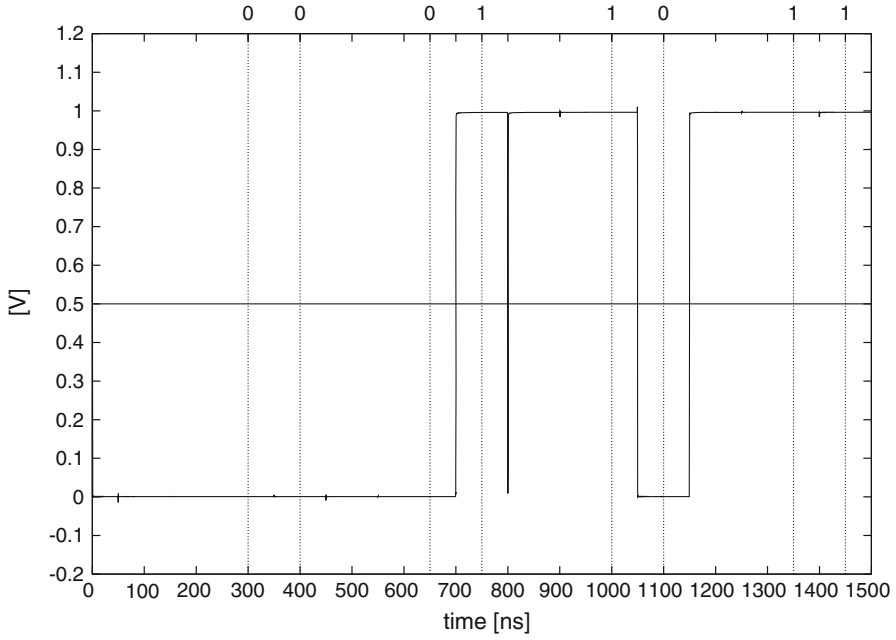


Fig. 11 Output of MSO LUT1, according to test vectors in Fig. 10. *Vertical dotted lines* indicate where the output is measured

When the test setup in Fig. 2 is applied for testing a LUT1, it is not possible to test all possible input values as was the case for the inverter experiment in Sect. 5.1 The fact that a LUT consists of storage elements means that a waveform must be employed, testing the effect of input vectors over time. A functionality test waveform for LUT1 is shown in Fig. 10. Figure 10 shows at which times the input signals are asserted and also shows the expected output value at different times. For calculating f_{rms}, the output is measured at the following times: 300, 400, 650, 750, 1,000, 1,100, 1,350 and 1,450 ms.

The output of a simulation of the LUT1 for the test vectors in Fig. 10 is shown in Fig. 11 and it can be seen that the LUT1 is functioning as intended. There is a glitch in the output at 800 ns which is the consequence of the SRAM cell delay. The delay

is visible as a glitch due to changing the "A" input at the same time as "W0" is asserted.

7 Evolved defect tolerant LUT

While the LUT1 in Sect. 6 is highly defect tolerant, the transistor count is very high and is thus not meeting our goal of area efficiency. As such, a second strategy towards a defect tolerant LUT is to directly evolve a LUT with redundancy. The hypothesis is that more efficient redundancy techniques can result if specialising towards the LUT functionality and letting evolution play with larger and more complex circuits.

7.1 Exploring experiment

A LUT1, even though much less complex than the more common 4-input LUTs, still presents a significant challenge to evolution. As such, the successful setup from Sect. 5.1 may not be suitable. An introductory and exploring experiment is, therefore, conducted where different experimental setups are tried. To keep complexity down on this introductory experiment, the only defects injected are gate-source shorts in nMOS transistors. The experiment in Sect. 5.1 showed that gate-source short tolerance can lead to tolerance also to other possible defect types. Concentrating on nMOS cuts the number of evaluations in half. In addition, the functionality test in Fig. 10 is reduced to the part between 450 and 1,150 ns.

It should be noted that the purpose of this experiment is not to conclude on the EAs ability to evolve circuits. Many more experiments are needed to draw any conclusion on such matters. Instead, the purpose is to generate at least one promising circuit to serve as a starting point for a refining experiment in Sect. 7.2.

Eight different experimental setups are tried and a total of 20 different evolutionary runs are conducted. The experimental setups differ in three areas: Seeding, elitism and test coverage. *Seeding* refers to how the initial population is created. The initial population is either completely random or seeded with five LUT1 circuits where nMOS transistors are made defect tolerant with the MSO technique from Sect. 5. The purpose of seeding is to optimise or enhance a given circuit, as opposed to starting evolving a circuit from just random individuals. *Elitism* is a feature of some EAs where the most fit individual is copied unaltered to the next generation. Elitism is either present or not. Elitism ensures that the fitness of the best individual in a population is never less than in the previous generation. *Test coverage* refers to how many of all possible single defects are tested during fitness evaluation. Complete test coverage provides the most accurate fitness estimation but is time consuming. Partial test coverage speeds up the evolutionary experiment. For runs with partial test coverage, the defects that are tested for are randomly selected.

For seeded experiments, the goal is to optimise the given redundant circuit and fitness function (5) is therefore applied. Non-seeded experiments start with a

Table 2 Results from the exploring LUT1 experiment

Run	Seed	Elitism	Test cov. (%)	f	R_{trad_single}	$\hat{f}_{rms}$	f_{rms}	Size	Works
	(Trad. designed)			0.998930	1.000000	0.996812	0.997838	165	Yes
1	Yes	No	100	0.998930	1.000000	0.996812	0.997838	165	Yes
2	Yes	No	100	0.993349	1.000000	0.996812	0.997838	165	Yes
3	Yes	Yes	100	0.998930	1.000000	0.996812	0.997838	165	Yes
4	Yes	Yes	100	0.998930	1.000000	0.996812	0.997838	165	Yes
5	Yes	Yes	10	0.998930	1.000000	0.996812	0.997838	165	Yes
6	Yes	Yes	10	0.988609	0.960396	0.986770	0.997712	158	Yes
7	Yes	Yes	10	0.998930	1.000000	0.996812	0.997838	165	Yes
8	Yes	No	10	0.998930	1.000000	0.996812	0.997838	165	Yes
9	Yes	No	10	0.976675	0.934783	0.980497	0.997325	143	Yes
10	Yes	No	10	0.998930	1.000000	0.996812	0.997838	165	Yes
11	No	No	100	0.443721	1.000000	0.609302	0.609302	4	No
12	No	No	100	0.425508	0.000000	0.551135	0.596305	3	No
13	No	Yes	100	0.871550	1.000000	0.673611	0.684732	8	Yes
14	No	Yes	100	0.415692	0.000000	0.502494	0.585967	4	No
15	No	Yes	10	0.424682	0.000000	0.537003	0.616108	4	No
16	No	Yes	10	0.424385	0.000000	0.552871	0.588953	3	No
17	No	Yes	10	0.419506	0.000000	0.534065	0.593164	4	No
18	No	No	10	0.433093	0.000000	0.585882	0.609286	4	No
19	No	No	10	0.421290	0.000000	0.503640	0.612811	4	No
20	No	No	10	0.427660	0.000000	0.518157	0.640041	5	No

random initial population and, therefore, needs fitness function (4) to provide sufficient information to evolve.

A genetic algorithm is applied with population size 20, mutation rate 0.02, crossover rate 0.2 and tournament selection with group size 3 and selection probability 0.7. The maximum number of transistors is 400 and the maximum number of nets is 407. Feedback is allowed to give evolution the possibility of evolving static storage elements. All other details of the experimental setup are as for the experiment in Sect. 5.1

Each evolutionary run is stopped after 1 week. Characteristics of the best individual from each run are shown in Table 2. In addition, the characteristics of the circuit applied as seed for the seeded experiments are shown in the first row of Table 2. The numbers in Table 2 are based on 100% test coverage, even for the individuals that were evolved with partial test coverage.

It is clear from Table 2 that the seeded experiments with complete test coverage did not manage to improve the seed in any way. The resulting individual has the same characteristics as the seeding individual. Some of the seeded experiments with reduced test coverage removed some redundant transistors, with the effect of reducing R_{trad_single}. This is probably due to the fact that reduced test coverage

results in a noisy fitness. "Noisy fitness" means that if fitness is evaluated twice for the exact same individual, the fitness value may vary. In this case, the reason is that a removed transistor may reduce the circuits ability to tolerate defects, but this is not necessarily detected by the fitness function as the circuit is not tested for all possible

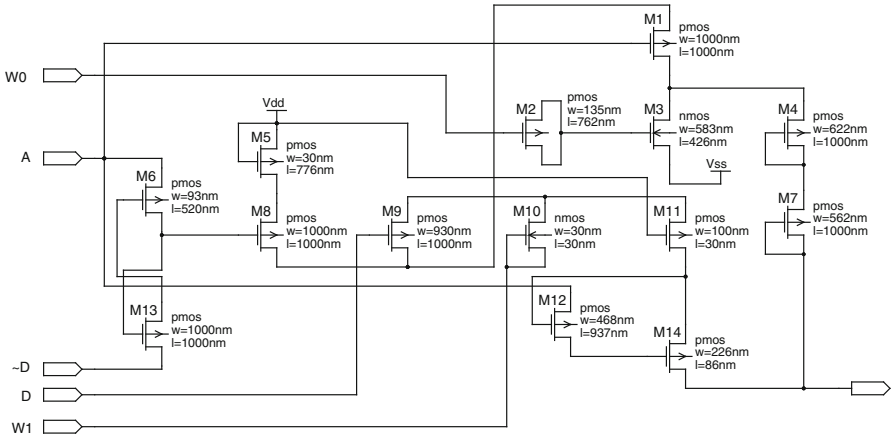


Fig. 12 Evolved LUT1

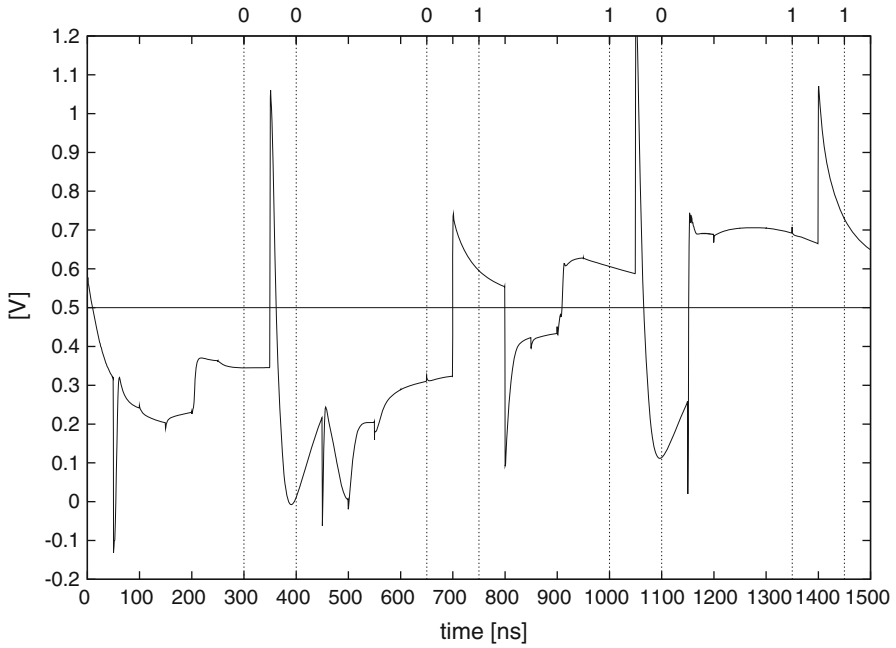


Fig. 13 Output of evolved LUT1, according to test vectors in Fig. 10. *Vertical dotted lines* indicate where the output is measured

defects. One interesting result is run 9 where the reduction in transistor count is larger than the reduction in R_{trad_single}.

For the non-seeded experiments, evolution was unable to find any working circuits, except in one case (run 13) with elitism and complete test coverage.

7.2 Refining experiment

The exploring experiment in Sect. 7.1 resulted in at least two interesting circuits. Run 13 resulted in a working circuit with complete tolerance to all single gate-source shorts in nMOS gates, yet consisting of only eight transistors. As the motivation for these experiments is a more area efficient redundant LUT than what was constructed in Sect. 6, run 13 was selected for a refining experiment.

The same experimental setup was applied, except with the full functionality test in Fig. 10 and gate-source short defects where injected in both nMOS and pMOS transistors. The initial population for the refining experiment was seeded with the individual from run 13.

As explained in Sect. 4.5, evolution was conducted in two phases. A parallel variant of the EHW simulator was run on 20 compute nodes on a cluster and each evolutionary phase was run for several days. Figure 12 shows the resulting evolved LUT1 and Fig. 13 shows a simulation for the test vectors in Fig. 10.

8 Discussion

To evaluate the two LUT1 implementations in Sects. 6 (MSO) and 7 (Evolved), two other LUT1 implementations have been constructed and simulated. The first is a traditional non-redundant LUT1 (Non-red.), as shown in Fig. 9. The second is a TMR implementation (TMR) where the non-redundant LUT1 is triplicated and a mirrored adder [14] applied as the voter. The mirrored adder has the very useful property of being tolerant to single stuck-closed and stuck-open defects when applied as a TMR voter, as long as there are no defective modules.

Characteristics for all four LUT1 implementations are given in Table 3 for comparison. R_{trad_trans} is estimated for a transistor reliability of 0.99 and is based on standard Monte Carlo simulations with 10,000 tests. Delay is the time the output needs for stabilising after the address input "A" changes. Delay measurements are

Table 3 Comparison of LUT1 implementations

Property	Non-red.	TMR	MSO	Evolved
Size	22 trans.	76 trans.	264 trans.	14 trans.
f_{rms}	0.998717	0.999346	0.997665	0.733561
$\hat{f}_{rms}$	0.779882	0.961079	0.994218	0.643702
R_{trad_single}	0.102273	0.858553	1.000000	0.446429
R_{trad_trans}	0.820000	0.882200	0.986700	0.921900
delay	< 1 ns	< 1 ns	< 1 ns	≈ 15 ns

based on the slowest transition for the test in Fig. 10 with no injected defects. Although not entirely accurate, these delay numbers provide an estimate for discussions. Timing requirements for writing to the LUT are not considered, based on the assumption that configuration of the LUT is rarely on the critical path for an FPGA application and, therefore, less important. The numbers in Table 3 are based on simulations where the full range of defect types are injected: gate-source short, gate-drain short, stuck-open and stuck-closed.

When considering R_{trad_single}, the MSO LUT1 is the most reliable and tolerates all possible single defective transistors ($R_{trad\_single} = 1$). The TMR LUT1 comes in second and tolerates 86% of all single defects. The reason TMR does not tolerate all possible single defects is a lack of tolerance to gate shorts. Some gate shorts in the TMR modules either pulls up or pulls down one of the module inputs so much that the other modules also fail. The voter also fails for some gate shorts.

For R_{trad_trans}, the MSO LUT1 is again the most reliable, estimated to 0.99. This means that given a transistor reliability of 0.99, the probability that the MSO LUT1 is working 100% is 0.99. TMR with 0.88 is a considerable improvement over the non-redundant LUT1 with 0.82. The evolved LUT1 is even better than TMR and is estimated to 0.92. The reason for the high R_{trad_trans} of the evolved solution despite the lower number of tolerated single defects, is the small size. The evolved LUT1 consists of only 14 transistors, resulting in a lower probability of having one or more defective transistors. 14 transistors is even less than the standard non-redundant LUT1, which is interesting, considering the higher number of tolerated single defects. The MSO LUT1 is by far the largest implementation in number of transistors, with 3.5 times the number of transistors of the TMR LUT1.

Although the evolved solution is both small and scores well on the reliability metrics, there are several disadvantages that separates it from all the three other LUT1 implementations. As evident from the simulation in Fig. 13, the evolved solution relies on some form of dynamic storage that is discharged by the output load. Some form of refresh is, therefore, needed. The output voltage swing is also very low, shown as a low value for f_{rms} in Table 3, and a restoring gate must, therefore, be present at the output. In addition, the evolved solution has a high delay.

This paper has concentrated on the 1-input LUT. Most FPGA LUTs today have from four to six inputs. The technique applied when constructing the LUT in Sect. 6 can easily be applied to any LUT implementation with more inputs. A multiple input LUT is, however, far too complex to be evolved with current EHW techniques and must therefore be constructed from several evolved 1-input LUTs and a defect tolerant multiplexer. Any general improvement on the scalability of evolvable hardware techniques will be of direct benefit for the techniques in this paper.

Some factors limit the accuracy of the results in Table 3. All simulations are based on the functionality test in Fig. 10 where only a limited number of test cases are present. It is possible that a faulty LUT may be classified as working because the tested input pattern does not reveal the faulty behaviour. Investigation into more effective ways to test a time dependent circuit, such as an LUT, is needed so as to reduce the evaluation time of a single individual.

In addition, the results are only valid for the four types of transistor defects considered in this paper. If other defects are considered, such as shorts in the interconnect between two transistors, the results could change. An important issue with scaled down devices is variability. Exploring solutions to such issues requires specialised models such as those being developed at the University of Glasgow, UK and incorporated in the EH defect tolerant work presented in [18]. The evolved circuits presented in this paper are not proposed as viable circuits but rather to highlight the advantages and disadvantages of the techniques at this stage. Further, refinement of the techniques and more accurate modelling of real world defects is needed before these techniques may be applied to evolve realistic solutions to real-world problems.

9 Conclusion

This paper has investigated the application of artificial evolution as a tool for achieving generic defect tolerant structures. Further, the challenges addressed and remaining are presented. An evolved defect tolerant structure (MSO) has been presented and applied to achieve a defect tolerant LUT for FPGAs and the resulting circuit has been compared to an evolved defect tolerant LUT in terms of defect tolerance and area usage.

Both the MSO LUT1 and the evolved LUT1 have advantages and disadvantages. The evolved solution is very small, yet still exhibits some tolerance to defects. As such, the evolved solution is an interesting example for further research. However, high delay; an output signal far from being close to perfect; possible input pattern ordering dependence and the fact that the LUT relies on dynamic storage, makes the evolved solution unrealistic in real FPGAs without further improvements. Further, the evolved solution is challenged with respect to evolvability due to the high evaluation cost.

The MSO LUT1 has the advantage of high output voltage swing, reasonable low delay and tolerates all single transistor defects of the four types this paper has addressed. Further, since the LUT design does not involve evolution but rather inclusion of the generic defect tolerant structure, scaling up the LUT will not be hindered by the complexity issues faced by the evolved solution. However, an unacceptably high area requirement requires further refinement of the technique.

References

1. Altera. Apex redundancy. http://www.altera.com/products/devices/apex/features/apx-redundancy.html
2. C. Bolchini, G. Buonanno, D. Sciuto, R. Stefanelli, A CMOS fault tolerant architecture for switch-level faults. in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pp. 10–18, 1994

3. C. Bolchini, G. Buonanno, D. Sciuto, R. Stefanelli, Static redundancy techniques for CMOS gates. in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 576–579, 1996
4. C. Bolchini, G. Buonanno, D. Sciuto, R. Stefanelli, An improved fault tolerant architecture at CMOS level. in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2737–2740, 1997
5. R.O. Canham, A.M. Tyrrell, Evolved fault tolerance in evolvable hardware. In *Congress on Evolutionary Computation (CEC)*, pp. 1267–1271, 2002
6. A. Djupdal, P.C. Haddow, Yield enhancing defect tolerance techniques for FPGAs. in *Military and Aerospace PLD International Conference (MAPLD)*, 2006. Paper ID 203
7. A. Djupdal, P.C. Haddow, Defect tolerant ganged CMOS minority gate. in *NORCHIP*, 2007
8. A. Djupdal, P.C. Haddow, Evolving efficient redundancy by exploiting the analogue nature of CMOS transistors. in *International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS)*, pp. 81–86, 2007
9. A. Djupdal, P.C. Haddow, Evolving redundant structures for reliable circuits—lessons learned. in *Adaptive Hardware and Systems*, pp. 455–462, 2007
10. A. Djupdal, P.C. Haddow, Defect tolerance inspired by artificial evolution. in *Proceedings of IEEE Ann. Symp. on VLSI*, 2008
11. A. Doumar, H. Ito, Design of switching blocks tolerating defects/faults in FPGA interconnection resources. in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pp. 134–142, 2000
12. M. Garvie, A. Thompson, Scrubbing away transients and jiggling around the permanent: Long survival of fpga systems through evolutionary self-repair. *On-Line Testing Symposium, IEEE International*, pp. 155–160, 2004
13. GEDA, Ngspice homepage. http://ngspice.sourceforge.net/, 2007
14. D. Hampel, K.J. Prost, N.R. Scheinberg, Threshold logic using complementary MOS device, June 1974. U.S. Patent 3 900 742
15. M. Hartmann, P.C. Haddow, Evolution of fault-tolerant and noise-robust digital designs. IEE Proc. Comput. Digit. Tech. **151**(4), 287–294 (2004)
16. 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, K. Kanzaki, Introducing redundancy in field programmable gate arrays. in *Proceedings of IEEE Custom Integrated Circuits Conference*, pp. 7.1.1–7.1.4, 1993
17. T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, T. Furuya, Evolving hardware with genetic learning: a first step towards building a darwin machine. in *From Animals to Animats: Simulation of Adaptive Behavior*, pp. 417–424, 1993
18. J.A. Hilder, J.A. Walker, A.M. Tyrrell, Optimising variability tolerant standard cell libraries. in *Proceedings of Congress on Evolutionary Computation*, pp. 2273–2280, 2009
19. ITRS, International technology roadmap for semiconductors. Technical report, ITRS, 2005
20. D. Keymeulen, R.S. Zebulum, Y. Jin, A. Stoica, Fault-tolerant evolvable hardware using field-programmable transistor arrays. IEEE Trans. Reliab. **49**(3), 305–316 (2000)
21. A.J. KleinOsowski, D.J. Lilja, The NanoBox project: Exploring fabrics of self-correcting logic blocks for high defect rate molecular device technologies. in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 1–10, 2004
22. I. Koren, Z. Koren, Defect tolerance in VLSI circuits: Techniques and yield analysis. Proc. IEEE **86**(9), 1819–1837 (1998)
23. P.J. Layzell, A. Thompson, Understanding inherent qualities of evolved circuits: Evolutionary history as a predictor of fault tolerance. in *International Conference on Evolvable Systems (ICES)*, pp. 133–144, 2000
24. R.E. Lyons, W. Vanderkulk, The use of triple-modular redundancy to improve computer reliability. IBM J. 200–209 (April 1962)
25. J.F. Miller, P. Thomson, Cartesian genetic programming. in *Genetic Programming, Proceedings of EuroGP*, pp. 121–132, 2000
26. M. Mishra, S.C. Goldstein, *Nano, Quantum and Molecular Computing, Implications to High Level Design and Validation, Chapter 3: Defect Tolerance at the End of the Roadmap* (Kluwer Academic Publishers, The Netherlands, 2004)
27. E.F. Moore, C.E. Shannon, Reliable circuits using less reliable relays. *J. Franklin Inst.*, pp. 191–208, 291–297, 1956
28. W.H. Pierce, *Failure-Tolerant Computer Design* (Academic Press, New York, 1965)

29. C.R. Saha, S.J. Bellis, A. Mathewson, E.M. Popovici, Performance enhancement defect tolerance in the cell matrix architecture. in *International Conference on Microelectronics*, pp. 777–780, 2004
30. A. Thompson, Evolving fault tolerant systems. in *Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, pp. 524–529, 1995
31. A. Thompson, Evolving inherently fault-tolerant systems. Proc. Inst. Mech. Eng. I J. Syst. Control Eng. **211**(5), 365–371 (1997)
32. K. Zhang, R.F. DeMara, C.A. Sharma, Consensus-based evaluation for fault isolation and on-line evolutionary regeneration. in *International Conference on Evolvable Systems (ICES)*, pp. 12–24, 2005
33. W. Zhao, Y. Cao, New generation of predictive technology model for sub-45 nm design exploration. in *International Symposium on Quality Electronic Design (ISQED)*, pp. 585–590, 2006