
Article:
Trefzer, Martin Albrecht orcid.org/0000-0002-6196-6832, Lawson, David Michael Renwick, Bale, Simon Jonathan et al. (2 more authors) (2017) Hierarchical Strategies for Efficient Fault Recovery on the Reconfigurable PAnDA Device. IEEE Transactions on Computers. 7756359. pp. 930-945. ISSN 0018-9340

https://doi.org/10.1109/TC.2016.2632722

Hierarchical Strategies for Efficient Fault Recovery on the Reconfigurable PAnDA Device

Martin A. Trefzer, *Senior Member, IEEE,* David M. R. Lawson, Simon J. Bale, James A. Walker, *Senior Member, IEEE,* and Andy M. Tyrrell, *Senior Member, IEEE*

Abstract—A novel hierarchical fault-tolerance methodology for reconfigurable devices is presented. A bespoke multi-reconfigurable FPGA architecture, the programmable analogue and digital array (PAnDA), is introduced allowing fine-grained reconfiguration beyond any other FPGA architecture currently in existence. Fault blind circuit repair strategies, which require no specific information of the nature or location of faults, are developed, exploiting architectural features of PAnDA. Two fault recovery techniques, stochastic and deterministic strategies, are proposed and results of each, as well as a comparison of the two, are presented. Both approaches are based on creating algorithms performing fine-grained hierarchical partial reconfiguration on faulty circuits in order to repair them. While the stochastic approach provides insights into feasibility of the method, the deterministic approach aims to generate optimal repair strategies for generic faults induced into a specific circuit. It is shown that both techniques successfully repair the benchmark circuits used after random faults are induced in random circuit locations, and the deterministic strategies are shown to operate efficiently and effectively after optimisation for a specific use case. The methods are shown to be generally applicable to any circuit on PAnDA, and to be straightforwardly customisable for any FPGA fabric providing some regularity and symmetry in its structure.

1 Introduction

The continuous scaling of transistors, now reaching 14 nm and below, has significantly increased the function density of modern digital systems and this has resulted not only in a significant reduction in the cost per logic function in an integrated circuit, but has also enabled unparalleled performance boosts with regards to computing power per Watt. Over the past 50 years, clock speeds of digital systems have increased from a few hundred kilohertz to the gigahertz regime and the number of transistors per die has increased from about a thousand to up to 6 billion. The fact that silicon die sizes have only doubled (or tripled at best) in the same time period, indicates a more than 1000-fold increase in device density.

While these numbers are staggeringly impressive, a consequence of fabricating transistors that small are structural irregularities at the atomic scale, even with advanced processes. For example, the presence or absence of single doping atoms affect device characteristics in random manner. While scaling transistors can reduce the propagation delay, power consumption and area of a device, this comes at the cost of increased intrinsic variability [1], [2] and heat dissipation. At the same time the significantly increased complexity of multi-billion-transistor devices makes designing a high speed, high yield digital system with low supply voltages and low power dissipation in ultra-deep sub-micron CMOS technology a major challenge. Successful design is only still feasible thanks to the existence of advanced electronic design automation (EDA) tools.

- *M. Trefzer, D. Lawson, S. Bale and A. Tyrrell are with the Department of Electronics, University of York, York, YO10 5DD, UK. E-mail: martin.trefzer@york.ac.uk*
- *J. Walker is with the School of Engineering and Computer Science, University of Hull, Hull, HU6 7RX, UK.*

Manuscript received August 22nd, 2016; revised November 13th, 2016; accepted November 21st, 2016.

While noise and device mismatch have always been present and have been posing major design challenges in electronic systems, three new challenges are coming together in modern electronic devices and systems making the requirement for cross-layer fault tolerance more paramount than ever [3]: (a) the stochastic nature of the variations at the nanoscale increasing reliability margins, (b) the drastic increase in the number of individual devices on a chip necessitating ever smaller per-device failure probabilities, and (c) ageing and wear-out becoming more rapid. As a result, faults and failure rates increase significantly and the impact of these low-level effects propagates from device level all the way up to the system level.

The greatest workload and responsibility remains to date with chip manufacturers, who continuously improve fabrication facilities and feed-back relevant design rules for creating the physical layout to the designers in order to ensure high yield figures. There are also certain post-fabrication measures that can improve the performance of a device or at least make it usable with reduced performance, for instance, altering power-supply voltages, slowing down clock-speed or disabling (redundant) parts.

However, when devices fail in fixed-function integrated circuits there is usually nothing that can be done to recover the functionality of a device. Even though the majority of components may be fault-free, the failure of a single transistor or connection will render at least part of a circuit permanently unusable. In contrast, reconfigurable devices, such as FPGAs, appear to open up more possibilities when dealing with faults. Theoretically, if a fault occurs on one part of an FPGA, it should be possible to perform a reconfiguration avoiding the faulty components. The challenge, however, is determining the appropriate reconfiguration to make as resynthesizing a design from scratch is computationally expensive, i.e. takes an infeasibly long time when multiple runs are required.

Many novel methodologies targeting fault tolerance using reconfigurable devices have been proposed in the literature. For example, [4] and [5] describe how partial bit strings could be made relocatable within a Xilinx FPGA, enabling a module to be moved to another place on the fabric in the event of a fault. Xilinx have looked into providing—on chip—a number of bit strings implementing the same functionality but are structurally different, so that an alternative could be loaded once a fault occurs. Closely related to this work, it was found in [6] that the tolerable failure rate of partially faulty LUTs in FPGAs could be significantly increased through the use of techniques such as permuting the inputs and changing their polarity to try to match faulty stuck-at outputs with the desired output. The techniques presented in [7] describe how partially faulty logic blocks allowed their system to even tolerate a greater number of faults, and a fine-grained approach using transistor-level reconfiguration for variability tolerance and yield improvement is discussed in [8], [9]. Another approach to increasing reliability and variability tolerance of programmable fabrics is presented in [10], where low-overhead circuitry is embedded assisting in on-line adaptation.

In addition to these pragmatic engineering approaches there have been methodologies developed in the field of bio-inspired hardware, which is where the work described in this paper is focussed. For instance, a successful concept developed is that of *embryonics*, which is inspired from organisms healing as a result of cellular mechanisms. The approaches developed in [11], [12], [13], [14] are modelling this kind of cellular healing mechanisms and relying on the presence of spare (hardware) cells that can be swapped into the running circuit in the event of a fault. Building upon the latter works, the Sabre platform has been developed [15], which is another cell-based system purpose-built for fault-tolerance. The fabric is made up of Functional Units (FUs) built from *Unitronics* cells [16]. An evolutionary optimsation approach to filter design can be found in [17]. Another evolutionary approach to increasing error resiliency of circuits, which is rooted on the field of approximate computing, is reported in [18].

In this work, we combine a bespoke bio-inspired multi-reconfigurable FPGA architecture [19], [20], [21]—the programmable analogue and digital array (PAnDA)—with novel "fault blind" circuit repair strategies taking inspiration from dynamic partial reconfiguration and configuration bit string permutation. In this case, "fault blind" refers to the proposed method's ability to repair faults without requiring information of the exact nature or location of a fault. This offers the advantage to fully concentrate on fault recovery and assume that fault detection in the granularity required here is readily available, and appropriate methods are proposed in [22], [23], [24].

This paper introduces two fault recovery techniques, *Stochastic Strategies* in Section 6 and *Deterministic Strategies* in Section 7. Both approaches are based on creating algorithms performing fine-grained hierarchical partial reconfiguration operations on a faulty circuit, on PAnDA, providing effective and fast fault recovery. While the stochastic approach provides insights into feasibility of the method, the deterministic approach aims to generate optimal repair algorithms (strategies) for generic random faults induced into a specific circuit. For optimisation of the strategies multi-objective optimisation is used [25], as multiple performance metrics are considered. The experiments conclude with a comparison of the two methods in Section 8.

2 PAnDA Architecture

The programmable analogue and digital array (PAnDA) architecture is a multi-reconfigurable fabric that consists of an array of configurable circuit blocks interconnected using a programmable routing structure. The term "multi-reconfigurable" refers to PAnDA's novel and unique feature to access its reconfiguration facilities on multiple design abstraction levels, each effectively representing a different granularity of the architecture. The highest configuration level makes PAnDA compatible to commercial FPGAs in the sense that logic functions can be mapped to configurable logic blocks (CLBs) that offer equivalent functionality and granularity: PAnDA Fünf slices (each CLB comprises two slices) can be configured as any 4-bit logic function, MUX or flip-flop, which is equivalent to the Xilinx Virtex-4 generation. In addition to that—and beyond the capabilities of any FPGA currently available—PAnDA can be configured on additional lower levels offering increasingly finer-grained configuration options all the way down to re-sizing individual transistors, which represent the lowest level of design. The routing architecture of PAnDA is currently following a standard approach comprised of switch matrices and a cross-bar architecture. Additional input/output (IO) blocks surround the CLB array and allow buffering of external signals.

PAnDA's multi-reconfigurability enables a wide range of capabilities and applications that are unique to this architecture. Consider the following examples: (i) A logic design mapped to PAnDA at the highest level can be optimised for better power/delay performance trade-off at runtime using transistor sizing. This effectively allows designers to modify aspects of the analogue circuitry underlying the digital function level. (ii) The operating point(s) of a design can be altered by increasing or decreasing underlying transistor sizes effectively shifting its performance characteristics as required. In addition, performance variations (probability distribution) caused by mismatch and intrinsic device variability, found in technologies below 100 nm, can be decreased by selecting an optimal set of devices from the alternatives available. (iii) Multi-granularity allows faults to be addressed and mitigated with the best cost/benefit trade-off, and exploiting symmetries of the architecture offers "fault-blind" repair capability by considering functionally equivalent but structurally different alternative mappings. We have previously shown that PAnDA can be used to provide increased circuit performance while simultaneously reducing the effects of variability using SPICE-level architecture simulation [19], [20]. Hierarchical strategies for fault tolerance in reconfigurable architectures, using PAnDA as a suitable candidate, is the subject of this work.

The PAnDA architecture has been developed in a number of iterations involving fabrication of prototypes in 65 nm silicon technology. The latest version is PAnDA-FÜNF, which features 400 CLBs and is a scaled-up version of its predecessor PAnDA-VIER. This sections provides details of

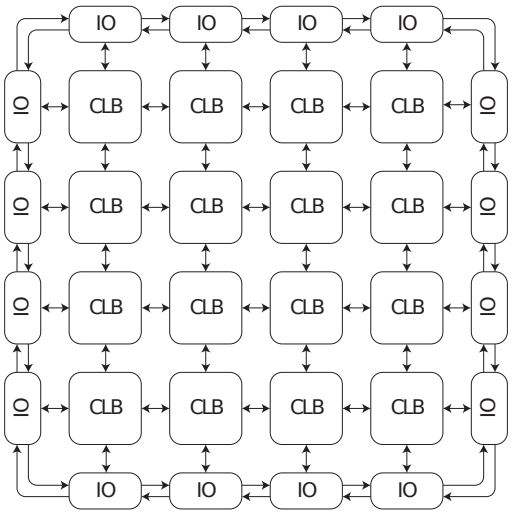

Fig. 1. The toplevel PAnDA architecture is shown. The array of 4×4 CLBs, surrounded by IO blocks, corresponds to PAnDA-VIER, which is a smaller version of PAnDA-FÜNF comprising 20×20 CLBs. Arrows signify programmable routing resources. A North-East-South-West (NEWS) routing block is part of every CLB and IO block.

the PAnDA-FÜNF architecture and its configuration options from the top down. The time required to compile a new bit-string and program a PAnDA-FÜNF chip is currently about $0.5 \ldots 1.0$ seconds.

2.1 Toplevel and Routing Block

The highest design level of PAnDA is the full chip (toplevel), shown in Figure 1. It consists of an array of configurable logic blocks (CLBs) comprising programmable logic resources (slices) and programmable routing, described in Section 2.2, surrounded by IO blocks. IO blocks are equivalent to the routing blocks contained within the CLBs, but take on the special role of routing signals to the pads (outside pins) of the device, rather than the neighbouring CLB.

2.2 Configurable Logic Block (CLB)

The CLB (Fig. 2) is a hierarchical structure consisting of two slices and a routing switch block. Each slice contains 4 configurable analogue blocks (CABs), input multiplexers and an output merger. Depending on the usage of a slice, i.e. how many CABs need to be connected via the output merger to create the desired logic function, the number of unique output signals varies from 1 to 4. The routing switch block provides external connectivity to the CLB array and allows up to 6 buses to be simultaneously routed north, east, west and south in both directions. In addition, any of the 24 incoming signals can be routed to both slices and the 8 outputs (4+4) of both slices can be routed to any of the 24 outgoing signals. Direct connections can be made within the routing block bypassing the slice logic. The 6 incoming signals on one side can be directly connected to the 6 outgoing signals on one of the three other sides in the order of the signals on the busses. At this level the CLB block is fully tileable making it in principle relatively easy to create larger array sizes.

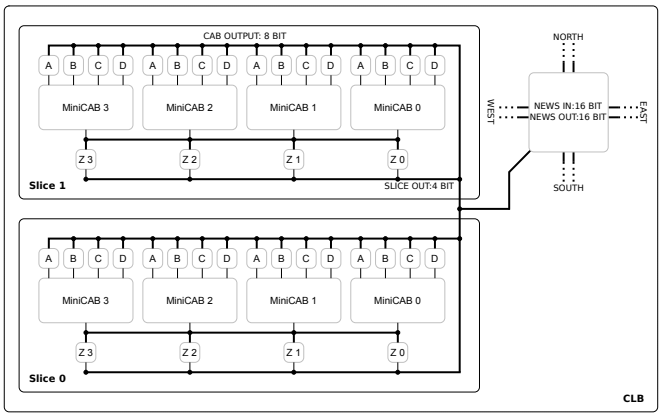

Fig. 2. A PAnDA configurable logic block (CLB) is shown. Each CLB comprises two slices and a routing block, and each slice consists of four configurable analogue blocks (CABs).

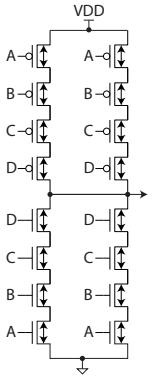

Fig. 3. The structure of the PAnDA configurable analogue block (CAB) is shown. Each CAB comprises 8 NMOS and 8 PMOS configurable transistors (CTs) arranged in two symmetrical branches.

2.3 Mini Configurable Analogue Block (MiniCAB)

Each CAB (Fig. 3) is constructed using 8 NMOS and 8 PMOS configurable transistors (CTs) arranged in two symmetrical branches. This branch structure closely matches the circuit topology seen in CMOS logic design and an arrangement of 4 CABs combined with an output merger allows any 4 input logic function to be implemented. Note that in the latest version of PAnDA CABs are also referred to as MiniCABs, however, this has only design-historical relevance and henceforth simply the name CAB is used here for simplicity.

2.4 Configurable Transistors (CTs): The Lowest Level of Configuration

Fig. 4 shows a schematic of a configurable transistor (CT). Each CT consists of six transistors, arranged in a parallel configuration, with gate widths either ranging from 135 nm to 230 nm, or all the same with the minimum-size gate width of 135 nm. The two resulting types of CTs are intended to serve different purposes: CTs with a variety of widths can achieve a greater number of combinations and maximum width (all turned on), which allows the manipulation of the *operating point* of a circuit. In contrast, CTs with devices of

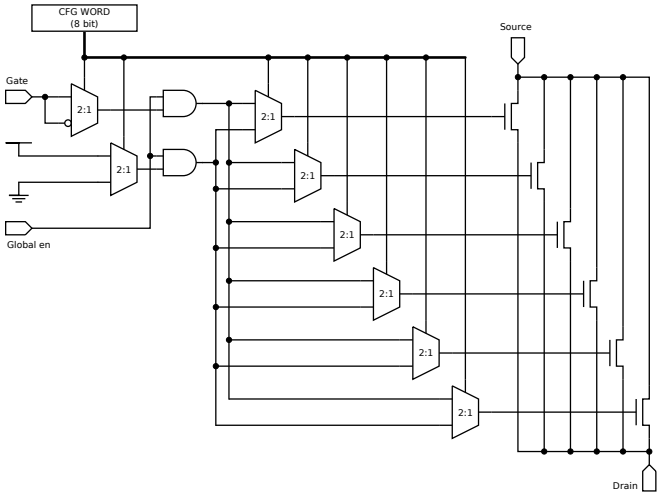

Fig. 4. The heart of PAnDA, a configurable transistor (CT), is shown. Each CT consists of six transistors, arranged in a parallel configuration, with gate widths ranging from 135 nm to 230 nm.

the same minimum width can achieve greater variability tolerance through offering more equivalently sized alternatives, i.e. allow to manipulate the *shape of the performance distribution*.

Configuration circuitry allows the input to the gates of each of these transistors to be enabled, disabled or clamped to a logic level. This allows any parallel arrangement of the six input transistors to be programmed resulting in an effective single transistor equivalent device with 64 possible gate width combinations ranging from 135 nm to 1045 nm (or from 135 nm to 810 nm).

3 Mapping Logic Functions

A PAnDA Slice (see Figure 2) is designed in a way that allows the implementation of any 4-bit logic function by following the implementation methodology of complementary MOS design where PMOS and NMOS transistors form a push-pull network driving an output. The PMOS branches are connected to VDD and will be driving the output 'high' for a logic '1', if *all* transistors in a branch are conducting. Vice versa, the NMOS branches are connected to GND and will drive the output 'low' for a logic '0', if *all* transistors are conducting. Notice that functions must be designed so that either (at least one) PMOS branch or NMOS branch is conducting. A PMOS and NMOS branch conducting at the same time causes a short circuit and neither a PMOS or NMOS branch conducting leaves the output undefined (floating).

Given a 4-bit input (ABCD), eight PMOS/NMOS branches are required to construct all possible logic combinations of ABCD and $\overline{\text{ABCD}}$, hence, a PAnDA slice provides eight branches. Note that the inverse signals are required, because PMOS and NMOS behave in the opposite (complementary) way where a logic '0' at the gate makes a PMOS conducting and an NMOS insulating, while a logic '1' makes a PMOS insulating and an NMOS conducting. While this design methodology allows the mapping of all possible 2^{16} boolean 4-input functions on a PAnDA slice, it will always consume all available resources. From a fault tolerance point of view this is undesirable, since this will leave no redundant CT's or branches that could be utilised in the event of a fault. However, the mapping of most logic functions can be optimised in a way as described in Appendix **??** that frees resources and thereby provides scope for the fault-mitigation strategies presented in this work.

This also ensures that all possible 4-bit boolean functions can be synthesised on PAnDA, as the mapping is different from traditional look-up-table (LUT) based FPGAs where a LUT is simply a memory and the inputs to the LUT are essentially the address bus "looking up" a corresponding output value. Hence, by storing a set of appropriate values in a LUT, it is possible to emulate any n-bit logic function, where n corresponds to the size of the address bus. PAnDA works differently in the sense that logic gates are implemented by configuring structures of transistors, as described above, which is more similar to how circuits are built in VLSI design.

4 Hierarchical Reconfiguration for Fault Mitigation

The PAnDA fabric offers a high degree of symmetry and homogeneity throughout its entire design hierarchy. It is hypothesised that the fabric's homogeneity can be utilised for the purpose of fault tolerance as every circuit can in principle be implemented in a number structurally different—but functionally equivalent—ways. Furthermore, it is suggested that symmetry can be exploited to apply certain non-disruptive circuit transformations which transform one possible implementation of a circuit into another, without any knowledge of its particular original mapping or structure required. When considering circuits that do not require all resources of a given area, i.e. there are spare CTs, branches, slices, CLBs etc. available, it follows that a circuit suffering from a fault within resources used might be transformed so that its new structure utilises intact, previously spare, resources, and the faulty components become the new, although possibly no longer useful, spares.

In fact, in all but two of the 2^{16} logic functions implementable on a PAnDA slice, there exists redundancy that can be exploited by way of an alternative implementation in order to recover from at least one type of fault in the reconfigurable fabric. The only two circuits that do not have this possibility are the most complex 4-input logic functions, odd and even 4-parity generators, when they are built as single logic gates. This is because these gates require all of the resources in a slice, leaving none spare for redundancy. Multiple implementations of these circuits are possible however and it seems likely that transistor reconfigurations at lower levels (inside a CT) could work around certain problems, although these will not be considered here as this work focuses on CT and branch level.

If a circuit implementation and the location of a fault occurring is known, and there are sufficient spares available, the circuit could certainly be fixed manually following the aforementioned methodology. However, when mapping a complex function and faults occurring at random locations at runtime, neither fault location nor positions of spares are known. Moreover, it is desirable to recover from a fault at runtime, without the need to perform a full reconfiguration

of the device or taking the entire system off-line. In order to address these issues and automate fault-mitigation a novel method, which requires no knowledge of the nature or the exact location of a fault, is introduced. It is based on algorithms that execute fault-blind repair strategies comprising of non-destructive circuit transformations. In this work we initially consider two reconfiguration hierarchy levels, input and branch swapping, however, the proposed approach is generic and can easily be extended across all hierarchies of PAnDA and it should be portable to any FPGA fabric offering a degree of symmetry and homogeneity, which they generally do. Moving up on the hierarchy, i.e. swapping entire CLBs, areas or even chips, requires a larger proportion of configuration bits to be moved around and potentially comes at a higher cost as larger areas become unusable spares, however, higher-level transformations will provide a higher probability to fix a fault. Therefore, finding strategies operating at the appropriate trade-off between fault-fixing performance and preserving resources becomes a multi-objective design problem.

4.1 Input Swapping

Within a CAB, the four inputs are each chosen from a set of 32. This allows a number of permutations for any given subset of inputs and the choice of permutation is functionally irrelevant, so long as the associated CTs are configured to match.

It can be seen therefore that the permutation of any Branch's input set can be changed to another, copying the CT configurations appropriately to match, and no knowledge of the current configuration is required as the resulting circuit will be functionally identical. This will be known as an Input Swap, and forms the lowest level transformation that this work considers.

Under certain conditions, Input Swapping allows fault recovery (see Figure 5). When a CT Branch has 1-3 CTs enabled, there are necessarily one or more CTs in the Disabled-Conducting state. Should any of these disabled CTs suffer a conducting fault, there will be no effect on the behaviour of the circuit, as they were already conducting. If a conducting fault occurs in an enabled CT, this clearly affects the behaviour of the circuit as the branch will conduct when it is not designed to. In this case, it is possible to take advantage of the previously mentioned case and swap the inputs and CTs around so that the *disabled-conducting* state configured for that CT is modelled by the fault and correct functionality is restored.

4.2 Branch Swapping

The Branch Swap transformation swaps Branches within a Slice. This is achieved by exchanging the configurations of all the CTs in a Branch with another. Branches can be swapped between CABs as well as within them while maintaining a functionally equivalent circuit. If branches are swapped between CABs, it is possible that the input multiplexers route the inputs in different orders. In this case, it is necessary to reorder the CT configurations as part of the swap. An example of a Branch swap is illustrated in Figure 5.

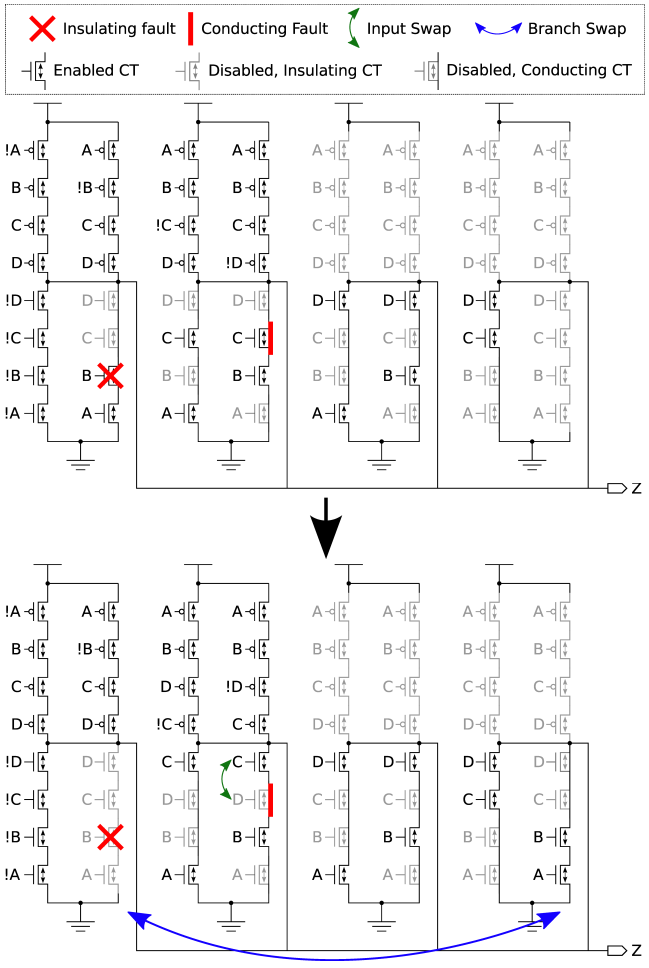

Fig. 5. An illustration of the application of two circuit transformations. The rightmost NMOS branch in the first CAB from the left suffers an insulating fault which is recovered from by being swapped with a spare in the fourth CAB. The rightmost NMOS branch in the second CAB from the left suffers a conducting fault, and is recovered from by swapping the mappings of inputs C and D.

If there is only one function currently occupying the Slice, all eight Branches of each type can by freely exchanged. If more than one function exists in a Slice however, the branches can only be swapped between CABs that form part of the same function. Although not explored here, the CABs could be swapped around so that each function utilises a different set of them, allowing the Branches to be swapped more freely when more than one function exists in a Slice, but this is not explored here.

5 Experiment Setup

This section describes the part of the experimental setup which is common to all experiments presented in this paper. This includes a description and discussion of the choice of benchmark circuits, of the fault model used and of the fault-injection process that has been followed when carrying out experiments in order to break circuit functionality.

5.1 Test Circuits

Four different circuits are used for testing which were chosen to represent a spread of different utilisations of branches

Test Circuit	Description
Z0	This circuit represents the simplest possible binary logic gate, an inverter. It utilises only one CT of a single branch of each type (PMOS/NMOS) in one CAB. Due to its simplicity, Z0 provides the highest possible amount of redundancy. All methods tested are expected to perform best on this circuit due to the maximum amount of spare resources available.
Z1	This is a four input "one hot detector" circuit, producing a high output if precisely one input is high. This circuit has been chosen as it provides a mix of redundancy between PMOS and NMOS branches. On the PMOS side, there are four fully occupied branches which means that input swapping will be ineffective, however, there are four completely redundant branches which will allow branch swapping to recover from faults. On the NMOS side, there is only one redundant branch, but all except one of the utilised branches have two unused CTs which will enable input swapping to recover from some faults.
Z2	This circuit has been chosen to represent a balanced mix of input and branch swapping possibilities on both PMOS and NMOS side. All except one of the branches has at least one spare CT, enabling fault recovery via input swapping, and there are four spare PMOS branches and three spare NMOS branches, allowing for fault recovery via branch swapping. It is expected that the proposed fault recovery methods will perform well on this circuit.
Z3	This circuit has a high utilisation, with most branches used having no redundant CTs, and there is only one redundant PMOS and NMOS branch respectively. This relatively small amount of redundancy does still provide some options for fault recovery, but it is expected to perform worse than Z0-Z3 in terms of the number of faults that it can tolerate and how long repair will take.

TABLE 1
The properties of the four benchmark circuits chosen as test cases for the proposed approach.

or slices. They are referred to as Z0 to Z3 for simplicity and consistency with other publications [26], [27], and their properties are listed in Table 1.

As pointed out in Section 4, four-bit even and odd parity circuits will not benefit from this methodology since they utilise all the CTs, leaving no spares. While this leaves no room for fault recovery within a single Slice, the functionality could be constructed by cascading two or three-bit parity circuits which do not utilise all the CTs. The 2-bit parity circuit fits inside a single CAB, and the three-bit inside two CABs. A four-bit parity circuit could be built with either a three-bit and a two-bit, or three two-bit circuits, either way taking up three CABs and leaving spare resources. If these were split up between different Slices then the fault tolerance methodology presented could be applied directly.

5.2 Fault Model

Based on the most common fault sources discussed in Appendix **??**, two types of faults are considered in this work: conducting and insulating transistor faults. These represent the two extreme cases of device failure and are commonly used in the literature as a basic fault model.

A conducting fault causes the faulty CT to permanently conduct between its source and drain nodes. This is suggestive of a short between the nodes, or perhaps the transistor suffering from variability and consequently having a very low or high threshold voltage, depending on whether it's NMOS or PMOS respectively. The functional effect of this is that the transistor conducts when not intended. In some situations, this will have no effect on the current configuration in PAnDA as CTs are sometimes used as conductors anyway. If this fault occurs in an active CT however, it will cause the CT to conduct on input patterns for which it is not supposed to. The effect of this is that for those input patterns, there may be a short circuit, for which there are two consequences. The first is that damage could be caused to the chip as a large current flows directly from Vdd to Ground, potentially damaging more transistors. The second is that, if the chip survives, the logic output will likely be either wrong or undefined ($\sim 0.5V$) due to the circuit essentially becoming a voltage divider. Since a CT suffering from a conducting fault acts as a conductor, performing the input swap transformation can recover from the effects of this fault by moving an input from a faulty CT to a working one, if the configuration allows.

Insulating faults prevent the transistor from conducting between source and drain. In reality this could be due to a break in the circuit due to electromigration or the threshold voltage being exceptionally high or low (for NMOS or PMOS transistors respectively). If this type of fault occurs on an unused CT branch, there will be no functional effect. If it occurs anywhere on an active branch, however, it will cause one or more input patterns to produce an undefined output, because the branch is physically cut off and cannot be used for anything else. This situation necessitates a branch swap transformation to move the configuration to another branch without an insulating fault.

As mentioned previously, each type of fault represents an extreme fault case and faults in reality may fall somewhere between the two, as well as manifesting in other ways (such as a transistor conducting between the gate and source or drain nodes). These other cases are not considered directly in this work but the methodology is still applicable since the precise nature of the fault is not considered during reconfiguration.

5.3 Fault Injection

Faulty circuits were created using the following process in the simulator:

1) The circuit was configured in a fault-free substrate.
2) The circuit was tested to ensure correct operation.
3) A fault of a random type (conducting or insulating) was injected into a random CT in the Slice.
4) The circuit was tested to see if the output is affected.
5) If the circuit still works the process is repeated from Step 3 and more faults are injected; if not, the process is finished.

Using this process, one or more faults are injected to break the functionality of the circuit. Faults which have no effect on the output are retained which may later affect the fault recovery. This was considered to be a realistic approach to modelling permanent faults - fault recovery would start after the first fault affects functionality, and faults not affecting functionality would accumulate unnoticed. It also makes

the fault recovery more challenging as the exact number of faults in any one case is unknown.

One observation of this approach is that a circuit with low utilisation, such as Z0, should on average receive more faults, and this may make fault recovery more difficult. This is because there is a large possibility that faults will be injected into CTs on unutilised branches, due to these forming the majority of the Slice configuration. These faults will accumulate until one of two things happens:

1) An insulating fault is injected into any of the CTs in the utilised branches.
2) A conducting fault is injected into either of the two active CTs.

The first situation will cut off the affected branch and cause it to always insulate. When the input pattern is such that the branch should conduct, the output of the circuit will float and likely cause an undesirable value to be read by whatever is connected to it.

The second situation will cause the affected branch to conduct for all input patterns. In the case of the inverter, this means that for half the input patterns the output will be correct, and for the other half the output will be connected to both VDD and GND and therefore be undefined.

Using a whole Slice to implement an inverter is quite inefficient as the PAnDA architecture allows for the CABs to be separated and used for separate functions, thus the three unused ones could implement other functions, but this design was chosen to measure the performance of the fault recovery in the best possible circumstances.

Figure 6 illustrates how many random faults need to be injected into each circuit to break their functionality. The numbers are expected and confirm that the choice of the four benchmark circuits is appropriate. The number of random faults that need to be injected into Z0 to break it is higher than for the other circuits. This is due to the majority of the slice being unused and so faults can accumulate in those parts without affecting functionality. Statistically, Z1 takes the second least number of faults to break, because there are four fully utilised branches (a quarter of the slice) any of which will be broken by a single fault. A slightly higher number of faults is required to break Z2, since most of the branches have at least one CT in the *disabled-conducting* state, they are able to absorb some conducting faults without functionality being affected. The least number of faults are required to break Z3. This is expected as most of the slice is utilised and so a random fault is likely to affect an *enabled* (used) CT.

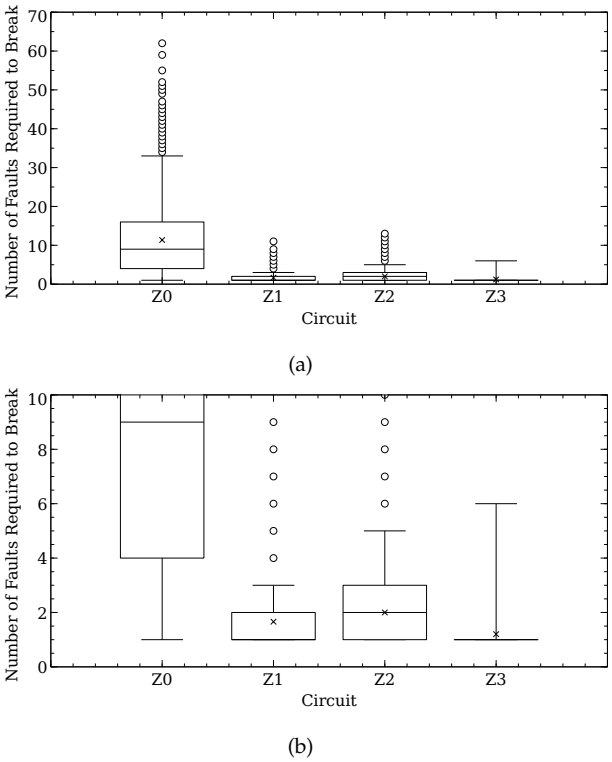

Fig. 6. Statistical illustration of how many random faults are required to break each of the circuits using box plots. The results were obtained by injecting faults into fault-free Slices configured with the functions until functionality was lost. This was repeated 10,000 times for each function. (a) The full results results showing the large difference between Z0 and the other circuits. (b) The full results truncated to show more detail of Z1, Z2 and Z3. The extremely skewed distribution in the case of Z3 comes from the fact that this circuit uses almost all resources available in a slice, which means that inducing just one fault already breaks it in most cases.

6 Stochastic Strategies

The two circuit transformations described in Section 4 were used as the basis for fault mitigation strategies. A sequence of one or more circuit transformations (reconfigurations) that an algorithm may apply to recover from a fault is referred to as a *strategy*. In the first set of experiments a set of strategies is defined, which is based on random selection. The stochastic approach serves two purposes: first, to demonstrate that the proposed approach is feasible and capable to repair circuits mapped onto PAnDA using a set of suitable reconfiguration operations. Second, the results from randomly applied strategies will provide a baseline comparison for the experiments and results in Section 7, where the sequence of strategies is optimised for a specific circuit. Another advantage of the stochastic strategies is that no prior knowledge of a circuit configuration is required as transformations are performed at random. In order to measure the effectiveness of these strategies in their ability to fix faults, experiments are conducted where the devised strategies are applied according to a random scheme to the benchmark circuits from Section 5.1 with faults injected, until correct operation is restored or a predetermined threshold is met.

There are 80 possible, distinct circuit transformations in total. Assuming that at least 10 transformations are required to fix a fault, which is a very conservative number given the fault-blind strategy, the size of the search space becomes 80^{10}. This is too large to enumerate and requires a search algorithm such as, for instance, an evolutionary algorithm as proposed here. Details on the combinatorial complexity of all circuit transformations can be found in Section 7, where it becomes more significant in the context of optimised deterministic strategies.

6.1 Fault Recovery Strategies

For these experiments four different strategies are devised, each using one of the two basic circuit transformations de-

scribed in Section 4: *Random Input Swapping*, *Input Shuffling*, *Random Branch Swapping* and *Branch Shuffling*. The strategies involving *swapping* are less disruptive than those performing *shuffling*, but the latter are capable of transforming large parts of a circuit more quickly.

6.1.1 Random Input Swapping

One of the CABs in a slice is randomly selected, and two of the four inputs (ABCD) are then swapped at random. This operation is relatively cheap and non-disruptive to the layout of the circuit, requiring the configurations of only two input multiplexers to be swapped and eight CTs (two on each branch of the CAB) to be reconfigured.

6.1.2 Input Shuffling

All inputs of all CABs in a slice are shuffled using the Fisher-Yates shuffling algorithm [28]. This, however, can be quite an expensive operation as the shuffle may require that all sixteen input multiplexers in a slice (inputs to all 16 CTs) are being reconfigured.

6.1.3 Random Branch Swapping

A branch type is first selected at random (PMOS or NMOS) and then two branches of that same type are then picked at random from the same slice and swapped. Again, this is a relatively cheap operation, requiring eight CT configurations to be exchanged (four for each branch). However, it can be disruptive to the layout of a circuit, causing a previously compact circuit to be spread out across a slice.

6.1.4 Branch Shuffling

All of the branches of both types are shuffled within a slice using the Fisher-Yates shuffling algorithm [28]. This is the most expensive transformation in terms of reconfiguration operations, potentially requiring all 64 CTs to be reconfigured. It is also the potentially most disruptive to the circuit layout.

6.2 Fault Recovery Method

In order to recover from a fault, the repair algorithm applies the strategies described in Section 6.1 to a circuit, once a fault has been detected, repeatedly and at random until either the circuit is fixed or a threshold of strategies applied (fault-fixing steps performed) is reached. Four experiments are conducted, testing each combination of one input swapping/shuffling and one branch swapping/shuffling strategy respectively:

- Random input swapping vs. random branch swapping
- Random input swapping vs. branch shuffling
- Input shuffling vs. random branch swapping
- Input shuffling vs. branch shuffling

6.3 Experimental Method

For each experiment, 101 runs are carried out, each with a different statistical bias, swept from 0 to 1 in 0.01 increments, making either input swapping/shuffling or branch swapping/shuffling more likely on average. For a bias of 0, only input-affecting strategies are used and for a bias of 1 only branch-affecting strategies are used. At a bias of 0.5, both input or branch strategies are applied with an equal probability.

It is assumed that the input-affecting strategies are generally preferable to the branch-affecting ones, as they cause less disruption to the mapping of a circuit. For example, an advantage of this is that small circuits, e.g. an inverter, does not quickly spread across multiple CABS. Moreover, branch swaps will be more complex due to the requirement to check—and possibly swap—the order of the CTs, if the input multiplexer configurations differ. Hence, when calculating the cost of an operation the power requirements are assumed to be higher for branch operations, despite the same number of CT reconfigurations are required. One of the aims of the experiments will hereby be to identify an optimum bias value, if it exists, which will represent good fault recovery performance while using branch strategies as little as possible.

For each of the 101 runs (representing 101 different bias values) of all four experiment setups, 10,000 tests are performed to achieve meaningful statistics. Preliminary testing resulted in this number of samples to be required for sufficiently stable results, whereas 1000 produced noticeable variance between repeats. The threshold for the maximum number of steps is set to 10,000, which allows the algorithm sufficient time to find a solution whilst not taking an infeasibly long time. In practice, this threshold will of course be too high for fault mitigation at runtime as it potentially involves tens of thousands of reconfigurations.

Each test run for the measurement of a bias value was conducted as follows:

1) Configuration of a function on a fault-free slice.
2) Fault-injection according to the procedure described in Section 5.3.
3) Start of the fault recovery procedure applying the strategies as described in Section 6.1.
4) Record the number of steps required to repair the circuit.

6.4 Results

Results of two performance aspects of the stochastic strategies applied to each of the four benchmark circuits Z0, Z1, Z2 and Z3 from Section 5.1 are presented: first, the total number of circuits repaired out of 10,000 instances. Second, the number of steps (strategies applied) required to recover from a fault in case repair has been successful. Results of an additional investigation into the average number of faults the benchmark circuits used can tolerate before they can no longer be fixed using the same approach are provided in Appendix **??**.

6.4.1 Number of Circuits Repaired

As can be seen from Figures 7(a)-7(d), the number of circuits repaired generally tends to increase as the bias favours branch swapping/shuffling strategies. When the bias reaches ~ 0.2, the number of circuits fixed becomes stable. In all four cases, the results for circuit Z2 are significantly higher than for the other three, averaging $\sim 9990 \pm 20$ in the stable section of the graph compared to $\sim 9800 \pm 50$

out of 10,000. It appears that the structure of Z2 is particularly suited to this method of fault recovery. The likely reason for this is that all but one of the active branches contain a spare CT, which provides a high probability of recovery through input swapping/shuffling when a conducting fault occurs in a CT. In addition, there are four spare PMOS branches and three spare NMOS branches, providing sufficient redundancy for branch swapping/shuffling to take advantage of.

In the way circuit Z2 benefits from redundancy, it may appear contradictory that circuit Z0, featuring the maximum amount of redundancy, does not. In fact for the majority of cases, fault recovery is much less successful on Z0 than any of the other circuits. The reason for this is likely down to the method of fault injection. Since faults are injected at random without regard for where the active CTs are placed, the Z0 circuit is more likely to receive a higher number of faults induced into unused CTs before finally breaking. Hence, when fault recovery is triggered, there are fewer fully functioning CTs left to work with. However, this reflects what would happen in reality and would be balanced by an average longer lifetime of Z0 before suffering from a fault.

The general shape of the graphs is as expected, since branch swapping/shuffling is principally able to recover from either type of fault (conducting or insulating), whereas input swapping/shuffling can only recover from conducting faults in branches which have a spare CT. However, as discussed previously, there is value in trying to use the input-affecting strategies as much as possible as the disruption to the layout of the circuit is significantly reduced. Hence, the best trade-off performance indicated by the results shown is around a bias of 0.2, where high success rates for fault recovery can be achieved while—on average—mainly using input swapping/shuffling.

At the far right of the graphs, the bias of 1 means that only use branch swapping/shuffling can be used. In this particular case the success rate drops dramatically, because there are specific combinations of faults from which it is only possible to recover with the use of input swapping/shuffling. For example, consider Z0 (see Table 1): if all the PMOS CTs in the top row are injected with conducting faults, branch swapping alone will not reach a working configuration again but an input swap moving the CT configuration down to one of the lower rows will. Therefore, despite branch swapping/shuffling strategies appear to be generally better than input swapping/shuffling strategies, the best performance is only achieved using a combination of both.

Figures 7(a) and 7(c) display worse performance at low values for the bias than Figures 7(b) and 7(d). The difference between these two sets is the branch strategy used. The experiments in Figures 7(b) and 7(d) use branch shuffling, which appears to have helped for bias values favouring input operations. Since the shuffling strategies can perform one or more swaps in one step, the total number of swaps performed in a single step is increased bringing the performance closer to that of the higher biases.

6.4.2 Number of Steps Taken

The mean number of steps required to fix a fault is shown in Figures 8(a)-8(d). As can be seen from the graphs, this number reduces as the bias increases towards using more branch swapping/shuffling, which means quicker fault recovery.

Comparing Figures 8(a) and 8(c) with 8(b) and 8(d), the mean number of steps towards the left hand side is lower in the latter two, where each step involves shuffling all inputs of all CABs. This enables the algorithm to fix any faults that are possible to repair with input swapping in a single step. However, due to the stochastic nature of how the strategies are applied, it is also possible to put the circuit into a situation where faults cause more problems than the previous state.

The results for Z2 are again noticeably better than for the other three circuits, which all perform similarly. The balance of redundancy and utilisation in this circuit resonate with this methodology well for two reasons: first, by utilising slightly more than half of the branches of a slice, faults are more likely to occur in utilised branches, leaving redundant ones less likely to be faulty and increasing the probability that spares will be fault-free when required. Second, having slightly less than half of the CAB branches not utilised leaves a lot of redundancy for branches to move into. Random swaps have a high probability of putting active branches into previously unused ones, speeding up fault recovery.

At the far right hand side of each of the graphs, performance is reduced in that the average number of steps required for repair increases. This indicates situations where it is hard for the algorithm to find configurations that work, and so more steps are required when trying to recover from faults. Reducing the bias slightly to include some input swapping clearly helps in these situations.

7 Deterministic Strategies

The experiments in Section 6 utilised a random application of repair strategies for fault tolerance and has been shown to successfully repair faults with varying levels of performance depending on the circuit it was applied to. It is therefore hypothesised here that an ordered application of deterministic strategies optimised for a given circuit should provide improved performance.

One drawback of the stochastic strategies approach is that there are frequently unnecessary swaps being performed that do not contribute to the repair process in a helpful way, which is expensive in terms of time and power due to making more reconfigurations. To a certain extent this is acceptable for a generic repair strategy capable of operating without knowledge of fault locations, however, with a completely non-deterministic approach the amount of unnecessary swapping is likely to be excessive. This should provide room for improvement when generating deterministic strategies tailored for specific use cases using an optimisation algorithm.

It is likely that, when the initial configuration of a circuit is known, similar strategies may be able to do better given some determinism. For instance, consider a circuit that is only using one PMOS and one NMOS branch, as is the case in Z0, where the majority of the branches are unused and therefore there is no effect in performing repair strategies on them. If the fault recovery method is deterministic and optimised for Z0, the algorithm will avoid swapping between

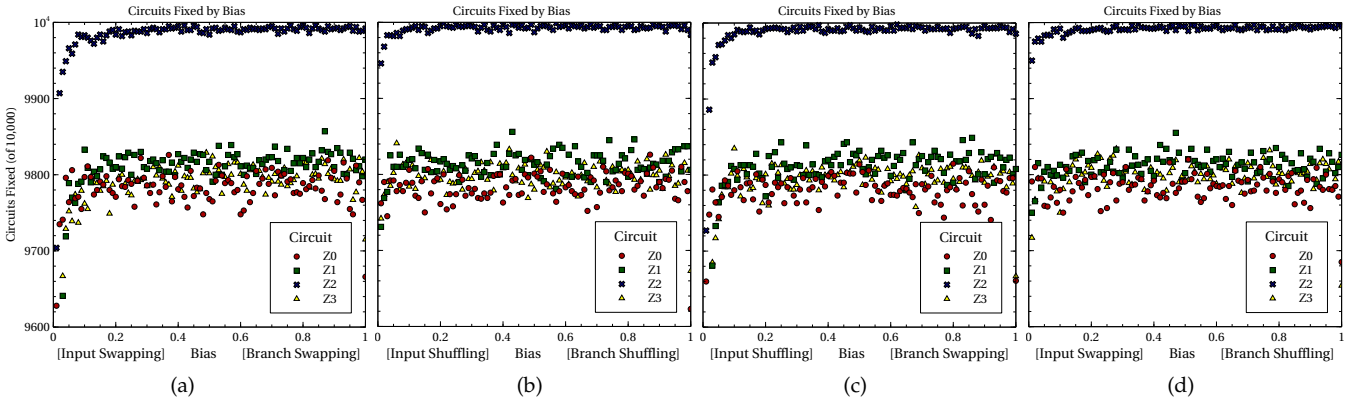

Fig. 7. The number of faulty circuits fixed is shown when running stochastic strategies sampling two different strategies with different statistical bias. The performance of input vs. branch swapping is shown in *(a)*, input vs. branch shuffling is shown in *(b)*, and the mixed experiments input shuffling vs. branch swapping and input swapping vs branch shuffling are shown in *(c)* and *(d)*. Generally, a trend towards better repair performance can be observed when branch operations are more likely used. However, the cost of these operations is also higher.

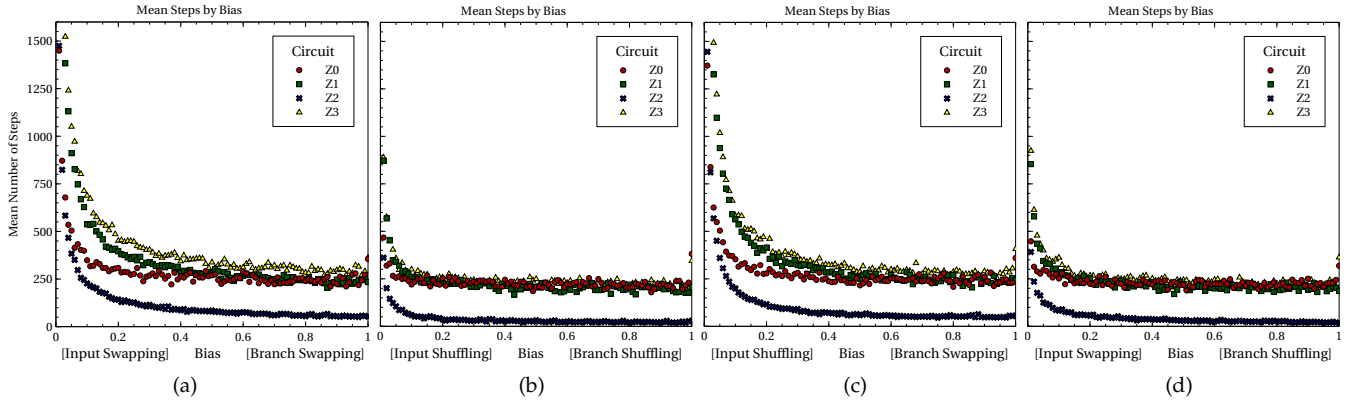

Fig. 8. The number of steps required (strategies applied) in order to repair a broken circuit is shown. The performance of input vs. branch swapping is shown in *(a)*, input vs. branch shuffling is shown in *(b)*, and the mixed experiments input shuffling vs. branch swapping and input swapping vs branch shuffling are shown in *(c)* and *(d)*. Again, a trend towards better repair performance can be observed when branch operations are more likely used. However, the cost of these operations is also higher.

two unused inputs or branches, resulting in faster runtime and less power consumed by unnecessary reconfigurations.

Therefore, in order to verify this hypothesis, experiments are conducted where deterministic strategies are optimised using a multi-objective evolutionary algorithm and are applied to the benchmark circuits from Section 5.1 with faults injected, until correct operation is restored or a predetermined threshold is met.

7.1 Fault Recovery Strategies

The rationale when determining the set of basic transformations is to keep the two types of strategy, input swapping and branch swapping, but to predetermine their application in some way. Where the strategies are previously composed of a random number of swaps, the new strategies are composed of exactly one deterministic swap, giving very fine grain control over what happens in the event of a fault. By enumerating all possible swaps within a slice, these swaps can be chained together to form a list of actions to perform, and the resulting efficiency of this list may be optimised.

The two types of circuit transformation described in Section 4 are again used to form strategies to repair faults. This time however, each strategy consists of just a single input or branch swap. It is calculated by enumerating all possible unique permutations that there are a total of 80 possible swaps that can be performed on a single slice, 24 input swaps and 56 branch swaps. These 80 swaps represent the set of operations to choose from when composing deterministic repair strategies. Due to the completeness of this set of transformations, shuffling is not used in the deterministic strategies.

7.1.1 Input Swapping

Each CAB has four inputs which results in there being six distinct swaps that can be performed between them. This can be calculated using the combination $\binom{4}{2} = 6$. Input A can be swapped with B, C and D. Input B can be swapped with C and D, since the A with B swap is already covered. Input C can be swapped with D and that covers all possible input swaps for one CAB. Since there are four CABs in a slice each with 6 possible swaps, there are 24 possible input swaps per slice.

Each swap is assigned an index in order to handle it conveniently in a repair algorithm. Input swaps are assigned the numbers 0-23, where 0-5 refer to swaps in the first CAB, 6-11 in the second and so on. Within each CAB, the swaps

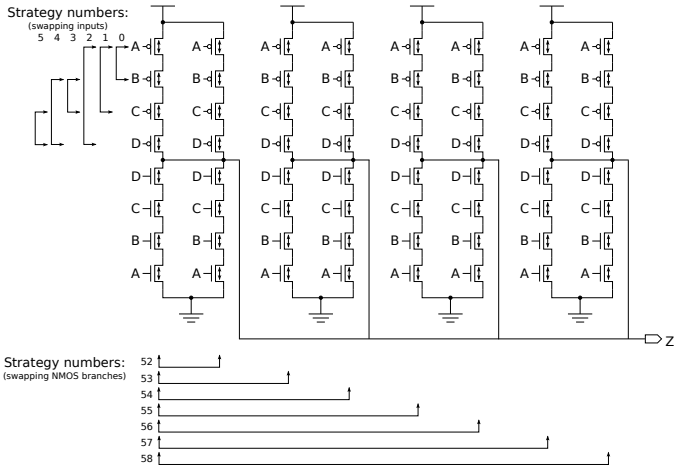

Fig. 9. An illustration of how some of the strategies are mapped from numbers to circuit transformations. For instance, strategy '0' swaps the input multiplexer configurations for inputs A and B, and also two CT configurations in each branch associated with inputs A and B. Strategies 6-11 perform the same actions on the second CAB from the left and so on. Strategy '53' swaps all four CT configurations in the leftmost NMOS CT branch with the ones in the third NMOS CT branch from the left, compensating for any differences in the order of the inputs between the two CABs. Strategies 59-64 swap the second NMOS branch with each of the others to the right and so on and strategies 24-51 enumerate all the possible PMOS CT branch swaps.

are arranged in the order used above, where the first is swapping A with B. Figure 9 illustrates the input swaps and identifiers assigned to them for the left-most CAB.

7.1.2 Branch Swapping

Each slice features sixteen branches, eight of each type (PMOS and NMOS). Only branches of the same type can be swapped, otherwise the functionality of the circuit changes after the swap. Consequently, the total number of possible swaps is $2 \times \binom{8}{2} = 56$.

The branch swaps are assigned index numbers in the same way as with the input swaps. PMOS branch swaps are numbered 24-51 and NMOS branch swaps 52-79. The first seven possible NMOS branch swaps are illustrated in Figure 9.

7.1.3 Strategy Lists

Associating every possible swap with a unique index number ensures than they can be referred to in a convenient and unambiguous manner. Arranging these numbers into a list (as in Figure 10) defines a deterministic set of actions that can be performed on a circuit. Lists of this type are henceforth referred to as strategy lists.

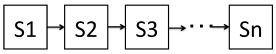

Fig. 10. An abstract strategy list. S{1-n} represent the strategy numbers which are applied in order.

7.2 Fault Recovery Method

Given a particular strategy list, the methodology when attempting to repair a fault works as follows:

1) Start with a faulty circuit with a known correct output.
2) Read the first number from the list.
3) Apply the strategy referred to by the number to the circuit.
4) Test the circuit to see if it now works.
    a) If it does, end the process.
    b) If it doesn't, read the next number from the list and continue from Step 3.

An example list, using numbers illustrated in Figure 9, could be [0, 3, 5, 52]. If the entire list is applied to the circuit Z0 from Section 5.1, the following circuit transformations will be performed:

1) Inputs A and B are swapped in the first CAB from the left.
2) Inputs B and C are swapped in the first CAB from the left.
3) Inputs C and D are swapped in the first CAB from the left.
4) The first and second NMOS branches from the left are swapped.

Note that it is assumed here that Z0 is mapped into the leftmost CAB of a PAnDA CLB. This is for illustration purposes only and not generally the case with a strategy list. The work presented here investigates whether it is possible to automatically create lists which repair faults in a more efficient—but still generic—way than the random method presented in Section 6.

7.3 Experimental Method

The proposed deterministic strategies methodology performs circuit transformations in a deterministic manner, specifically single-swap strategies stored in an ordered list. While the stochastic strategies apply a different sequence of circuit transformations each time it is used, the deterministic method applies the same sequence of transformations each time. The experiments performed in this section are aimed to automatically generate and optimise an appropriate sequence of strategies, i.e. a strategies list, for repairing faults. In order to allow a comparison with the stochastic method, the same set of benchmark circuits Z0-Z3 is used here.

The length of the lists used in the experiments is 50, i.e. a maximum of 50 strategies are applied when attempting to recover from a fault. This value, which is significantly lower than the 10,000 used in the stochastic method, has been chosen based on the results from Section 6.4, where circuit Z2 requires approximately 50 steps on average to repair and the others take significantly more than that. Calibrated to this critical point, the results from this section should therefore show clearly any advantage the deterministic approach may have.

7.3.1 Multi-objective Evolution of Strategy Lists

The process of creating a strategy list for a given function should be automatic. However, the search space is infeasibly large for manual exploration and its structure is unsuitable for hill-climbing. Hence, to achieve this automation, an evolutionary algorithm (EA) is employed which optimises

lists iteratively. Moreover, multiple goals for the optimisation, e.g. fixing as many faults as possible whilst requiring the least number of steps, necessitate the use of a multi-objective evolutionary algorithm (MOEA), for which NSGA-II is used. In this case, the generation, optimisation and evaluation is performed using the commercial optimiser LS-OPT [29]. The algorithm used is LS-OPTs implementation of NSGA-II.

The genetic encoding simply consists of 50 variable placeholders representing a generic list of 50 integer numbers forming a strategy list. Each number can take on a value between 0 and 79, each representing a strategy from the basic set of transformations described in Section 7.1.3. The EA parameters used are:

- Population: 80
- Mutation rate: 0.1
- Generations: 400

Population size and mutation rate are the defaults suggested in LS-OPT. The experiments are stopped after a limit of 400 generations has been reached, as preliminary experiments showed that by then solutions have already seen significant convergence and further improvement has slowed down.

7.3.2 Objectives

The following objectives are identified to be minimised by the MOEA:

1) Number of unfixed circuits (out of 1,000).
2) Average number of steps required for repair.
3) Cost/effort expended to fix circuits.

Each genome (strategy list) is evaluated on 1,000 instances of the same circuit with different faults randomly injected. The number of unfixed circuits value represents the number of faulty circuits that are not fixed out of these 1,000.

In addition to the two measurements used Section 6, a cost metric is introduced. The purpose of this is to encourage the optimisation algorithm to favour input swapping over branch swapping. The rationale behind this is that if a fault can be fixed with just input swapping, it will disrupt the layout of the circuit significantly less than branch swapping. A solution achieving this is clearly superior.

The input swapping strategies are assigned a low cost of 1, whereas the branch swapping strategies are assigned a high cost of 100. The optimisation algorithm is instructed to minimise the total cost expended when executing a certain list. The specific cost values are arbitrarily chosen in a way to significantly penalise the branch swapping and encourage input swapping.

7.3.3 Strategy List Evaluation

The process of evaluating a strategy list is repeated 1,000 times for each list. A more detailed explanation follows:

1) A fault-free PAnDA Slice model is prepared and programmed deterministically with a logic function.
2) The circuit is made faulty by the process described in Section 5.3.
3) The algorithm looks to the first item in the strategy list.
4) The strategy stored at the current position of the strategy list is applied to the circuit.
5) The circuit is checked for correct functionality.
6) If it is not fixed and there are items left in the list, the algorithm looks to the next item in the strategy list and goes back to Step 4. If it is fixed or every item in the list has been used, this evaluation is over and the number of steps taken are added to a running total.

As in the previous experiments in Section 6, each of the 1,000 evaluations of the list has to deal with a random number of randomly injected faults, in order to prevent over training to any particular set of faults.

7.4 Results

Results for each of the four functions Z0-Z3 are shown in Figure 11. These solutions represent the Pareto-optimal set found after running the MOEA for 400 generations. As can be seen from the figure, the general trend seen in all four cases is that solutions leaving fewer circuits unfixed—i.e. achieve to repair more—do so, on average, in fewer steps. However, this comes at the expense of cost, which can be seen increasing as the other two objectives decrease. This suggests that, for faults that are indeed repairable, the evolved strategy lists are able to repair them quickly. At the same time, these lists generally comprise a large amount of branch swapping operations.

In order to illustrate a typical result of the evolutionary algorithm, the first eight steps of an "unfixed"-optimised list for Z0 is worked through and explained in more detail in Appendix **??**.

8 Comparison of Stochastic and Deterministic Strategies

In order to provide a more direct and fair comparison between the deterministic and stochastic approaches discussed in Sections 7 and 6, results presented here are obtained from a slightly modified version of the experiment using the stochastic strategies.

8.1 Modifications to the Setup

There were two different input swapping methods used in the stochastic strategies: a single random input swap and a shuffle of all the inputs in the slice. The first mechanism is equivalent to the one used for the deterministic strategies, apart from swaps being randomly selected in the first case and running one of the strategies numbered 0-23 in the second case. However, the equivalent of the shuffle mechanism would involve running up to three of these on each of the CABs, so up to 18 single swap strategies in total for one input shuffle operation. To provide a more fair comparison, this strategy was modified to make a single swap on a single CAB by invoking a random deterministic strategy in the range 0 to 23.

The experiments in Section 6 do not consider cost, as this is not meaningful when sweeping the bias value. However,

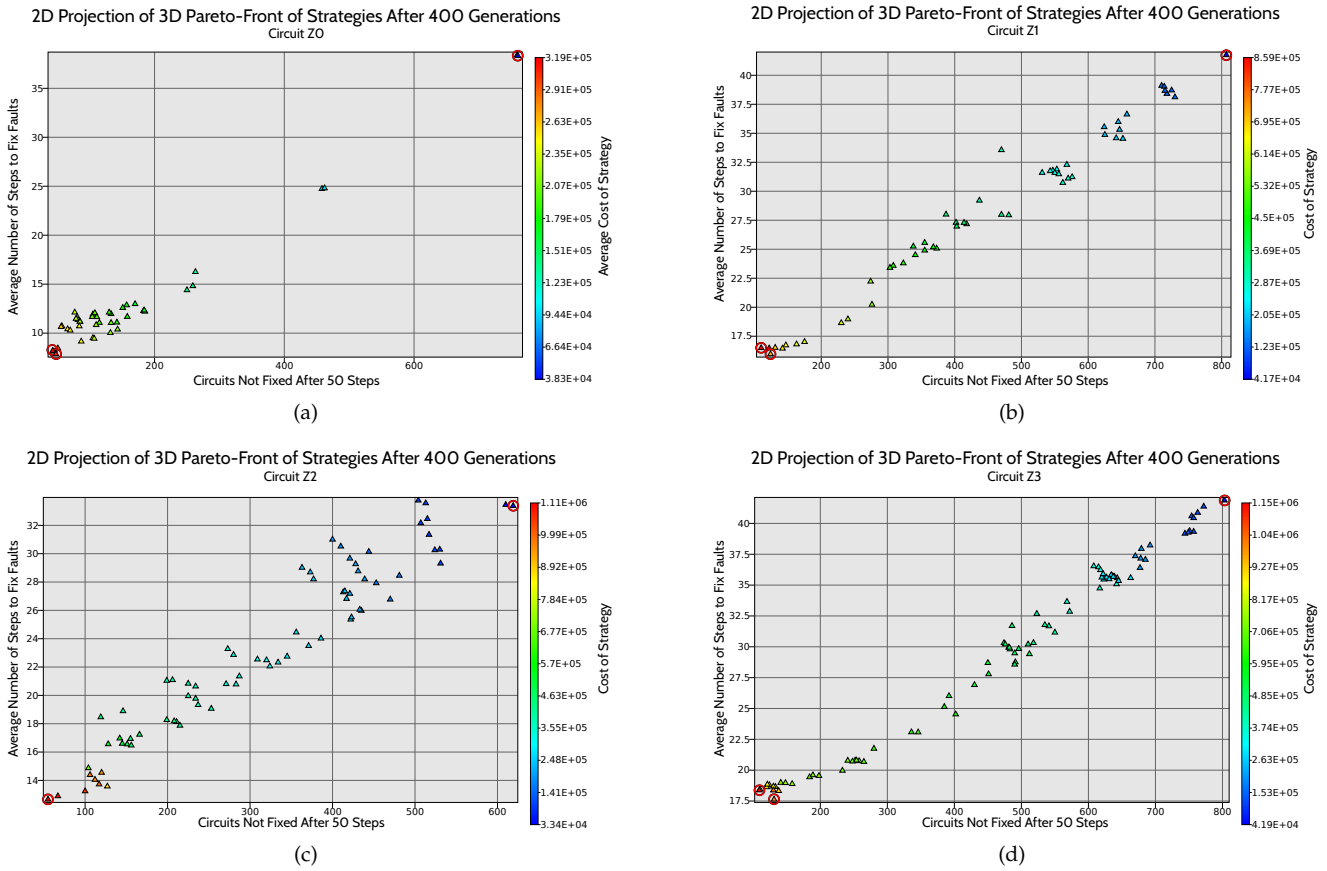

Fig. 11. The results for circuit Z0 is shown in *(a)*, Z1 is shown in *(b)*, Z2 is shown in *(c)* and Z3 is shown in *(d)* after evolving their respective strategy lists over 400 generations. The best solutions optimised for the number of unfixed circuits, mean number of steps and total cost are circled in red.

this metric was added simply for comparison with the total cost being accumulated over all (successful and unsuccessful) runs.

The measurement of the mean number of steps to fix a fault is modified to fit with the optimisation-based approach of the deterministic strategies. Instead of recording the number of steps taken in only the successful cases, the total number of steps used over all runs is used in order to remove statistical bias. Finally, The total number of steps is divided by 1,000 to provide an average per run.

8.2 Results

For comparison, the results from the stochastic method from Section 6 are re-run with modifications detailed in Section 8.1 on the same four benchmark circuits used before. The new results are shown in Figure 12.

The general trend that can be seen from Figures 12(a) and 12(b) is that the average number of steps required to recover from a fault and the number of circuits left unfixed decrease as the bias tends towards the branch swapping strategy. Figure 12(c) shows the opposite trend in that the average cost decreases as the bias tends towards the input swapping strategy. This is intuitively expected as the branch swapping strategy is significantly more expensive than the input swapping strategy but it is also expected that there would be more of an 'S'-shaped curve as the introduction of branch swapping should overall decrease the number of strategies that have to be applied. It is possible, however, that this effect becomes visible with more carefully selected cost factors.

TABLE 2
A comparison of solutions from both methods optimised for the least unfixed circuits out of 1000.

Circuit		Solutions Optimised for Least Unfixed		
		Stochastic	Deterministic	Improvement
Z0	Unfixed	113	43	61.95%
	Steps	16.973	8.2	51.69%
	Cost	1629485	317575	80.51%
Z1	Unfixed	256	110	57.03%
	Steps	24.571	16.487	32.90%
	Cost	2457100	858779	65.05%
Z2	Unfixed	150	55	63.33%
	Steps	19.918	12.655	36.46%
	Cost	1850032	1106210	40.21%
Z3	Unfixed	383	109	71.54%
	Steps	31.991	18.374	42.57%
	Cost	3135740	1148560	63.37%

The number of circuits repaired in the best cases of each approach is significantly improved, as can be seen from Table 2. The improvement varies between 57.03% and 71.54% in terms of the number of circuits repaired, and the other objectives are improved as well. This confirms that for the given number of 50 steps, the deterministic method is indeed able to recover more circuits from faults.

The deterministic method has decreased the average

TABLE 3
A comparison of solutions from both methods optimised for the least number of steps taken during 1000 circuit repairs.

Circuit		Soolutions Optimised for Least Steps Stochastic	Deterministic	Improvement
Z0	Unfixed	122	48	60.66%
	Steps	16.44	7.859	52.20%
	Cost	1644000	311987	81.02%
Z1	Unfixed	256	124	51.56%
	Steps	24.571	15.961	35.04%
	Cost	2457100	720742	70.67%
Z2	Unfixed	150	55	63.33%
	Steps	19.918	12.655	36.46%
	Cost	1850032	1106210	40.21%
Z3	Unfixed	395	131	66.84%
	Steps	31.73	17.627	44.45%
	Cost	3173000	992381	68.72%

TABLE 4
A comparison of solutions from both methods optimised for the least cost expended after 1000 circuit repairs.

Circuit		Solutions Optimised for Least Cost Stochastic	Deterministic	Improvement
Z0	Unfixed	793	760	4.16%
	Steps	41.798	38.309	8.35%
	Cost	41798	38309	8.35%
Z1	Unfixed	835	807	3.35%
	Steps	43.778	41.745	4.64%
	Cost	43778	41745	4.64%
Z2	Unfixed	688	619	10.03%
	Steps	39.826	33.375	16.2%
	Cost	39826	33375	16.2%
Z3	Unfixed	855	803	6.08%
	Steps	45.398	41.863	7.79%
	Cost	45398	41863	7.79%

number of steps taken when fixing faults by between 35.04% and 52.2% (see Table 3). This suggests that, if a fault is repairable, the deterministic method will find the fix quicker, on average.

The new cost measurement shows less improvement than the other optimised solutions (see Table 4). The deterministic solution presented shows a small improvement in cost, between 4.64% and 16.2%, and similarly sized improvements across the other objectives.

9 Conclusion

This work has demonstrated how symmetry-aware mapping of logic functions onto FPGA architectures in combination with evolutionary algorithms can be a powerful tool devising fault-blind repair strategies using existing reconfiguration mechanisms. This paper has focussed on fault-mitigation in the case of transistor failures. Some of the PAnDA architecture features, symmetries and swapping of inputs, have been proven to be useful for fault-blind fault mitigation. The approach presented here can be generalised to any existing FPGA architecture that exhibits regularities and symmetries in its architecture. Future work will be considering the more unique features of PAnDA

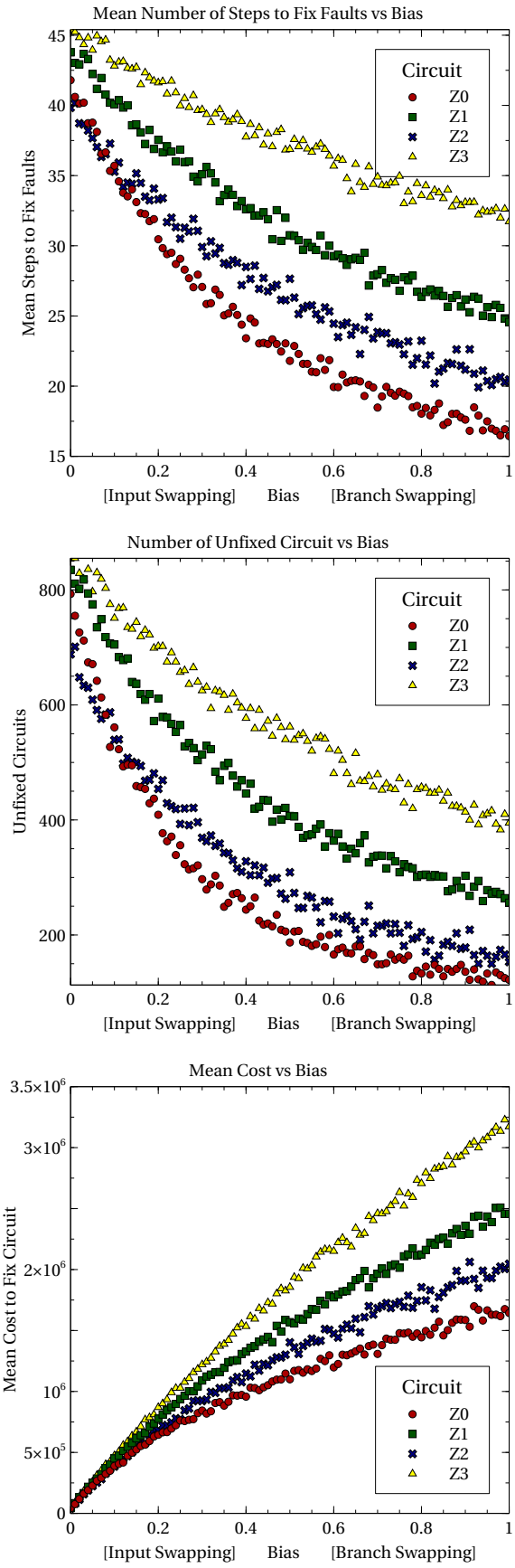

Fig. 12. Results of running the experiments from Section 6 with the differences detailed in Section 8.1. *Top:* the average number of steps required to fix random faults. *Middle:* the number of unfixed circuits out of 1,000. *Bottom:* the total cost expended when using a particular bias.

that go beyond what currently available FPGAs can do, which is reconfiguration at the transistor level, in order to extend hierarchical fault-mitigation to this lower level and to expand the scope of fault tolerance towards design optimisation. For example, configurable transistors do not only allow fixing faults, but also offer a range of possibilities to optimise a mapped design for performance, variability or power consumption by changing and optimising transistor sizes.

The results of the *Stochastic Strategy* experiments from Section 6 have shown that even with *random* circuit transformations, it is possible to structurally manipulate circuits on PAnDA-FÜNF in a way to work around faults. The methodology described has been demonstrated to work without any information about the configured circuit's implementation or the location or nature of any faults, simply the knowledge that a circuit is faulty. By exploiting the additional configuration layers available on PAnDA, fault recovery has been performed in a simple yet effective way. It has also been shown that the efficiency and effectiveness of the *Stochastic Strategy* methodology can be controlled by biasing the random application of transformations between two different strategies. This effect can be used to optimise the methodology for a particular situation. For example, if the quickest possible recovery is required, biasing towards branch swapping/shuffling will be optimal, whereas if the disruption to the layout of a circuit should be kept minimal, biasing towards the input swapping/shuffling side will be more beneficial while yielding equally good results.

The experiments using *Stochastic Strategies* found that both types of transformations, input and branch based ones, are required to achieve the best results. If biased strongly towards the input swapping strategy there is a drop in performance that becomes severe at the extreme. When biased towards the branch swapping strategy, the results remain good until very high values of bias, at which point there is a significant (but not nearly as severe) drop in performance. Some circuits performed better than others in the same circumstances. This is due to differences in their layouts changing the probability or difficulty of finding recovery configurations.

The results of the *Deterministic Strategy* experiments have shown that this method is able to perform better than the stochastic approach in all test cases. The results suggest that this would be the case for every possible function; also this follows from the fact that the stochastic method is likely to perform many unnecessary and unhelpful reconfigurations, which will not be the case if the strategies are optimised and more deterministic ensuring a more consistent reconfiguration path to a successful repair.

The *Deterministic Strategy* experiments have also shown that applying an ordered list of strategies provides a better performing fault tolerance methodology than a random application, even when the source of the fault is unknown. The results have also shown that it is possible to optimise the time it takes to fix faults, or trade this off for lower structural disruption to the circuit configuration. A slight drawback here is the time required to evolve good strategy lists, which makes it infeasible to produce them on the fly at runtime. However, strategy lists can be computed in advance, i.e. after circuit mapping and before deployment, and stored in a memory so that an optimal strategy is available in the event of any random occurring fault at runtime.

ACKNOWLEDGMENTS

This work is part of the PAnDA project that is funded by EPSRC (EP/I005838/1) and the EPSRC Platform Grant: Bio-inspired Adaptive Architectures and Systems (EP/K040820/1).

REFERENCES

[1] A. Asenov, "Random dopant induced threshold voltage lowering and fluctuations in sub-0.1um MOSFET's: A 3-D "atomistic" simulation study," *IEEE Transactions on Electron Devices*, vol. 45, no. 12, pp. 2505–2513, 1998.

[2] A. Asenov, S. Kaya, and A. Brown, "Intrinsic parameter fluctuations in decananometer MOSFETs introduced by gate line edge roughness," *Electron Devices, IEEE Transactions on*, vol. 50, no. 5, pp. 1254–1260, 2003.

[3] ESIA, JEITA, KSIA, TSIA, and SIA, "International Technology Roadmap for Semiconductors 2.0 (ITRS)," Industry Association, Tech. Rep., 2015. [Online]. Available: http://www.semiconductors.org/main/2015_international_technology_roadmap_for_semiconductors_itrs/

[4] D. Montminy, R. Baldwin, P. Williams, and B. Mullins, "Using relocatable bitstreams for fault tolerance," in *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, Aug. 2007, pp. 701–708.

[5] W.-J. Huang and E. J. McCluskey, "Column-based precompiled configuration techniques for FPGA," in *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, Mar. 2001, pp. 137–146.

[6] A. DeHon and N. Mehta, "Exploiting partially defective LUTs: Why you don't need perfect fabrication," in *Field-Programmable Technology (FPT), 2013 International Conference on*, Dec. 2013, pp. 12–19.

[7] J. Emmert, C. Stroud, and M. Abramovici, "Online Fault Tolerance for FPGA Logic Blocks," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 2, pp. 216–226, Feb. 2007.

[8] P. Wilson and R. Wilcock, "Yield improvement using configurable analogue transistors (CATs)," *Electronics Letters*, vol. 44, no. 19, pp. 1132–1134, 2008.

[9] P. R. Wilson and R. Wilcock, "Optimal sizing of configurable devices to reduce variability in integrated circuits," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2009, pp. 1385–1390.

[10] H. S. Low, D. Shang, F. Xia, and A. Yakovlev, "Asynchronously assisted FPGA for variability," in *24th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2014, pp. 1–4.

[11] G. Tempesti, D. Mange, A. Stauffer, and C. Teuscher, "The biowall: an electronic tissue for prototyping bio-inspired systems," in *Evolvable Hardware, 2002. Proceedings. NASA/DoD Conference on*, 2002, pp. 221–230.

[12] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti, "Toward robust integrated circuits: The embryonics approach," *Proceedings of the IEEE*, vol. 88, no. 4, pp. 516–543, Apr. 2000.

[13] M. Boesen and J. Madsen, "edna: A bio-inspired reconfigurable hardware cell architecture supporting self-organisation and self-healing," in *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, Jul. 2009, pp. 147–154.

[14] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, and S. Durand, "Embryonics: A new family of coarse-grained field-programmable gate array with self-repair and self-reproducing properties," *Towards evolvable hardware*, pp. 197–220, 1996.

[15] M. Samie, G. Dragffy, A. M. Tyrrell, A. G. Pipe, and P. Bremner, "A novel bio-inspired approach for fault-tolerant vlsi systems," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 21, no. 10, pp. 1878–1891, October 2013.

[16] M. Samie, G. Dragffy, and T. Pipe, "Unitronics: A novel bio-inspired fault tolerant cellular system," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, Jun. 2011, pp. 58–65.

[17] M. Murakawa, S. Yoshizawa, T. Adachi, S. Suzuki, K. Takasuka, M. Iwata, and T. Higuchi, "Analogue EHW chip for intermediate frequency filters," in *From Biology to Hardware: International Conference on Evolvable Systems (ICES)*, vol. 1478. Springer, LNCS, 1998, pp. 134–143.
[18] Z. Vašíček and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.
[19] M. A. Trefzer, J. A. Walker, S. J. Bale, and A. M. Tyrrell, "Fighting stochastic variability in a D-type flip-flop with transistor-level reconfiguration," *IET Computers and Digital Techniques*, vol. 9, pp. 190–196, 2015.
[20] J. A. Walker, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "PAnDA: a reconfigurable architecture that adapts to physical substrate variations," *IEEE Transactions on Computers*, vol. 62, no. 8, pp. 1584–1596, 2013.
[21] M. A. Trefzer, J. A. Walker, and A. M. Tyrrell, "A Programmable Analog and Digital Array for Bio-inspired Electronic Design Optimization at Nano-scale Silicon Technology Nodes," in *IEEE Asilomar Conference on Signals, Systems, and Computers*, Nov. 2011.
[22] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, Dec. 2003, pp. 7–18.
[23] N. Naber, T. Getz, Y. Kim, and J. Petrosky, "Real-time fault detection and diagnostics using FPGA-based architectures," in *2010 International Conference on Field Programmable Logic and Applications*, Aug. 2010, pp. 346–351.
[24] K. Inoue, Y. Nishitani, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "Fault detection and avoidance of FPGA in various granularities," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012, pp. 539–542.
[25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[26] D. Lawson, J. Walker, M. Trefzer, S. Bale, and A. Tyrrell, "A hierarchical fault tolerant system on the panda device with low disruption," in *Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on*, Jul. 2014, pp. 69–76.
[27] ——, "Evolving hierarchical low disruption fault tolerance strategies for a novel programmable device," in *Evolvable Systems (ICES), 2014 IEEE International Conference on*, Dec. 2014, pp. 77–84.
[28] R. A. Fisher and F. Yates, *Statistical Tables for Biological, Agricultural and Medical Research*. London: Oliver & Boyd, 1938.
[29] N. Stander, W. Roux, A. Basudhar, T. Eggleston, T. Goel, and K. Craig, *LS-OPT®User's Manual: A Design Optimization and Probabilistic Analysis Tool for the Engineering Analyst, Version 5.0*, Livermore Software Technology Corporation, 2013.

Dr Martin A. Trefzer studied Physics at the University of Heidelberg, Germany, and at the Technische Universität Berlin, Germany, from which he received a 1^{st} class honours degree. He obtained his PhD in Physics from the Kirchhoff-Institute for Physics, University of Heidelberg, Germany, where he was working on the evolution of transistor circuits using a custom designed CMOS FPTA. After his PhD he started working on artificial developmental hardware systems as a Research Associate at the Department of Electronics, University of York, UK. He is now a Lecturer in the Department of Electronics, University of York, UK. He is a Senior member of the IEEE.

David M. R. Lawson David M. R. Lawson has an MEng degree from the University of York, United Kingdom in 2011. He is currently working on a PhD at the Department of Electronics at the University of York as part of the EPSRC funded PAnDA: Programmable Analogue and Digital Array project. His research interests include evolutionary algorithms, reconfigurable architectures and fault tolerance.

Dr Simon J. Bale obtained his M.Eng. and Ph.D. degrees from the University of York, UK in 2004 and 2012 respectively. He is currently employed as a Research Associate in the Department of Electronics at the University of York on the EPSRC funded PAnDA: Programmable Analogue and Digital Array project. In the evolutionary computing area, his research interests include: microelectronic design optimisation using bio-inspired methods and techniques. In the RF/microwave area, his research interests include: high Q microwave resonators, ultra low phase noise oscillators and low noise measurement techniques. He is a member of the IEEE.

Dr James A. Walker is a Research Fellow in the Department of Electronics at the University of York on the EPSRC funded 'PAnDA: Programmable Analogue and Digital Array' project. He was previously a Research Associate on the EPSRC funded e-science pilot project 'Meeting the Design Challenges of Next Generation Nano-CMOS Electronics' in the same department. He received his PhD in Electronic Engineering from the University of York in 2007, for which he was awarded the Kathleen Mary Stott Memorial Prize for Excellence in Scientific Research. He received his BSc in Mathematics and Computer Science and MSc in Advanced Computer Science from the University of Birmingham in 2002 and 2003 respectively. He is a Senior member of the IEEE.

Prof. Andy M. Tyrrell received a 1st class honours degree in 1982 and a PhD in 1985 (Aston University), both in Electrical and Electronic Engineering. He joined the Department of Electronics at the University of York in April 1990, he was promoted to the Chair of Digital Electronics in 1998. Previous to that he was a Senior Lecturer at Coventry Polytechnic. Between August 1987 and August 1988 he was visiting research fellow at Ecole Polytechnic Lausanne Switzerland, where he was researching into the evaluation and performance of multiprocessor systems. From September 1973 to September 1979 he worked for STC at Paignton Devon, on the design and development of high frequency devices. He is a Senior member of the IEEE and a Fellow of the IET.