2013 IEEE Congress on Evolutionary Computation
June 20-23, Cancún, México

Overcoming Faults using Evolution on the PAnDA Architecture

Pedro B. Campos, David M. R. Lawson,
Simon J. Bale, James Alfred Walker, Martin A. Trefzer and Andy M. Tyrrell
Intelligent Systems Group, Department of Electronics, University of York, Heslington, York, YO10 5DD, UK
Email: {pbc500, dl520, simon.bale, james.walker, martin.trefzer, andy.tyrrell}@york.ac.uk

***Abstract*—This paper explores the potential for transistor level fault tolerance on a new Programmable Analogue and Digital Array (PAnDA) architecture[1]. In particular, this architecture features Combinatorial Configurable Analogue Blocks (CCABs) that can implement a number of combinatorial functions similar to FPGAs. In addition, PAnDA allows one to reconfigure features of the underlying analogue layer. In PAnDA-EINS, the functions that the CCAB can implement are predefined through the use of a routing block. This paper is a study of whether removing this routing block and allowing direct control of the transistors provides benefits for fault tolerance. Experiments are conducted in two stages. In the first stage, a logic function is evolved on a CCAB and then optimised using a GA. A fault is then injected into the substrate, breaking the logic function. The second stage of the experiment consists of evolving the logic function again on the faulty substrate. The results of these experiments show that the removal of the routing block from the CCAB is beneficial for fault tolerance.**

I. Introduction

Electronic circuits are not immune to faults. There are numerous different ways in which a particular circuit can develop a fault, especially with stochastic transistor variability becoming an issue in small manufacturing processes [1]. Taking a single transistor as an example, its behaviour can vary from its ideal characteristics significantly. It can fail completely and simply become a permanent conductor or insulator, or it can fail somewhere in-between such as the threshold voltage being too high or too low. Failing completely is more commonly known in the digital domain as a “stuck-at” fault, as in a stuck-at-1 or a stuck-at-0 fault. This failure mode is examined at the analogue level.

PAnDA is a novel reconfigurable architecture that aims to include reconfigurable circuit structures at different levels of abstraction/complexity [2]. This paper considers the architecture from the first and second iterations of the PAnDA chip and makes a comparison between the two. Its various levels of reconfiguration lend themselves well to research in transistor variability and fault-tolerance, as well as being a versatile substrate for general evolvable hardware study. This work focuses on using structures from PAnDA as a substrate for evolvable hardware, and using that to examine the possibility of developing fault tolerant systems at the transistor level. All experiments are carried out in simulation.

[1]This work is funded by EPSRC under the PAnDA project (EP/I005838/1)

II. Background

This work comes as a result of bringing together many different areas of electronic engineering. Fault tolerance is a large area in the research community but due to limitations on space, only a few of the most relevant topics are presented below.

A. Evolvable Hardware

The Heidelberg and JPL FPTA work [3][4] demonstrates that it is feasible to evolve circuits at the transistor level. Langeheine [5] and Trefzer [6] demonstrated this by evolving a number of analogue and digital circuits on an array of transistors. The CCAB used in this work has a much smaller search space than the FPTA, so intuitively circuits should be found quicker.

B. Fault-Tolerant Design

Fault tolerance is an important topic in engineering as a whole, but especially in electronic engineering. Triple Modular Redundancy (TMR) is a commonly used approach in critical systems. It uses modular redundancy and a voting system to provide fault tolerance. TMR relies on the voting system being fault free. Although there is redundancy in the number of transistors used in this work, it is not a fixed factor like in TMR, and being a reconfigurable architecture rather than application specific, the transistors aren’t always redundant. Currently however, our experimental system does rely on a separate, working system to recognise faults and evolve a working function again.

C. Evolved Fault Tolerance

Evolutionary techniques have previously been used for fault tolerance [7][8][9][10]. Canham used an FPGA as a substrate to evolve simple oscillator circuits in two environments - one which was fault free and one with injected faults. It was found that the circuits evolved in the environment with injected faults showed a 12.5 times increase in fault tolerance to the circuits evolved in the environment without faults.

D. Optimisation for Variability

Hilder [11][12] found that certain combinations of widths lead to an increase in tolerance to transistor variability related issues. The method used in Hilder’s work involved the use of evolutionary algorithms to find these combinations in a

TABLE I
THE 16 CONFIGURABLE FUNCTIONS OF THE PANDA CCAB.

Configuration	Function (Standard Output)	Function (Inverted Output)
0	Inverter	Buffer
1	NAND-2	AND-2
2	NOR-2	OR-2
3	NAND-3	AND-3
4	NOR-3	OR-3
5	AND-OR-INV-21	AND-OR-21
6	OR-AND-INV-21	OR-AND-21
7	PROG-AND-OR-INV-2	PROG-AND-OR-2

similar way, although one difference in this work is that we are changing the effective widths of our Configurable Transistors (CTs) instead of actually changing transistor widths. These experiments are more limited due to the fact that there is a finite set of widths which can be selected in a CT, whereas Hilder's work did not have this constraint.

III. PANDA ARCHITECTURE

PAnDA is an architecture that is reconfigurable on multiple levels [13]. At the highest level of PAnDA-EINS[2] are Configurable Logic Blocks (CLBs), which are comparable to FPGA blocks in that they can be configured to implement different combinatorial or sequential logic functions, and two 8-bit inputs. CLBs consist of eight Configurable Analogue Blocks (CABs), four each of two types: Combinatorial CABs (CCABs) and Sequential CABs (SCABs), which can both be configured to perform one of eight gate level functions, with the inverted output giving a total of sixteen (See Table I for the functions of a CCAB). Each CAB consists of fourteen CTs, which behave as single NMOS or PMOS transistors that can have their properties configured via the changing of their effective widths.

A. Configurable Transistors

The characteristics of an ideal CMOS transistor are defined, in part, by its physical width. In general, wider transistors tend to perform faster, but consume more power, and vice-versa. The trade-off that a particular circuit will require depends on the application. Transistor variability introduces additional levels of complexity as "fault" modes, although study of this will be left for future work. PAnDA CTs have a configurable width (120nm to 1140nm), post-fabrication, and so can be generalised as being single transistors that can be resized.

As in regular CMOS transistors, CTs have three main nodes: gate, source and drain. Inside a CT (see Fig. 1), there are seven CMOS transistors connected in parallel, with the sources and drains all connected together and the gates connected together through configurable switches. These switches have the effect of varying the effective width of the CT by either enabling or disabling transistors.

[2]The first version of the PAnDA architecture.

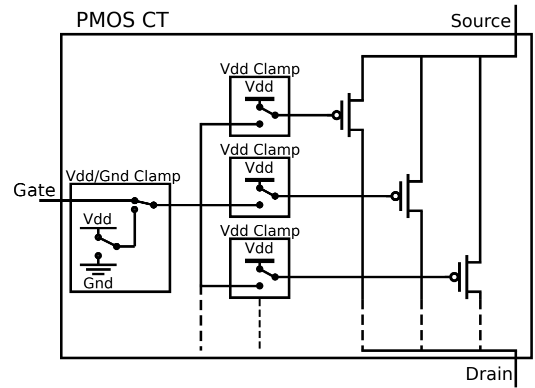


Fig. 1. The structure of a PMOS flavour CT showing three of the seven transistors. The "Vdd/Gnd" clamp controls whether the whole CT is enabled, insulating or conducting, and the "Vdd" clamps control the effective width.

TABLE II
CT CONFIGURATIONS

Enable	Clamp Configuration	PMOS CT State	NMOS CT State
0	0	Insulating	Conducting
0	1	Conducting	Insulating
1	0	Enabled	Enabled
1	1	Enabled	Enabled

Enabling a CT refers to the gate node being attached to a functional input (A, B or C) of the respective CAB it is part of. Disabling a transistor refers to connecting the transistor's gate to either Vdd (1V in this case) or Ground (0V), which makes the transistor unconditionally either insulate or conduct. Therefore, each transistor can be in one of three states: **enabled**, **conducting** or **insulating**. Configurations are shown in Table II.

For the experiments shown, our CTs are configured by nine bits. There are two to configure the clamps as in Table II and seven more to configure the width of the CT. When configuring the width, transistors are disabled by making them insulate. This means that a CT can also be put into an insulating state by setting its width to zero.

B. PAnDA-ZWEI CCAB

The PAnDA-ZWEI[3] CCAB (see Fig. 2) architecture allows functions to be configured on it by enabling and disabling specific combinations of the CTs. In PAnDA-EINS, a decoder and routing block use three configuration bits to configure one of eight predefined functions. This work is a study of whether removing the routing block and decoder used in PAnDA-EINS and adding in configurable clamps to CTs benefits fault tolerance.

C. PAnDA Fault Tolerance

The structural elements on the PAnDA fabric make it a suitable substrate for evolvable hardware, such as the inclusion of Configurable Transistors (CTs) and configurable clamps. The Combinatorial Configurable Analogue Block (CCAB) –

[3]The second version of the PAnDA architecture.

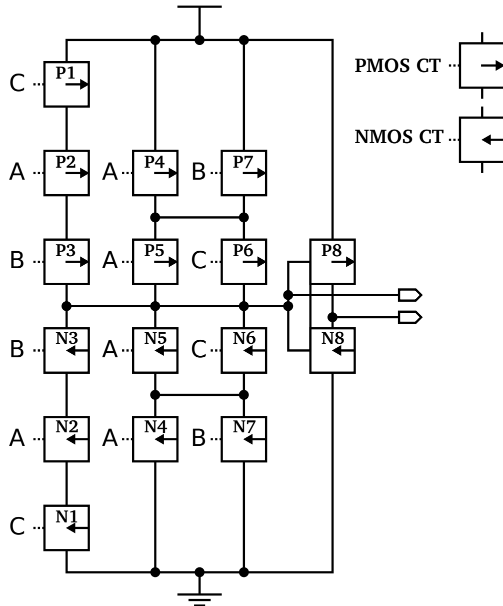


Fig. 2. The structure of the PAnDA-ZWEI CCAB showing the layout of the PMOS and NMOS transistors. The letters A, B and C indicate which of the three CCAB inputs goes to the gate of each transistor. Note that the rightmost PMOS and NMOS transistors form an inverter to generate the complementary output.

which is the fundamental combinatorial building block in PAnDA– is made up of CTs and configurable clamps and has a high level of flexibility, since the functional structure can be completely changed by loading new configurations onto the clamps.

There are multiple ways of implementing some functions on a CCAB, giving room for faulty transistors to be worked around. Below that, because of the nature of the CTs, faulty transistors within them can be disabled in some circumstances, giving redundancy at the lowest level.

Using the CT and CCAB structures from our PAnDA-EINS chip, experiments were carried out to see whether, should a transistor develop a fault in a working circuit, evolution could find a way to recover functionality. As an example, in the CCAB structure, there are tens of ways of building an inverter (using different pairs of PMOS and NMOS CTs), so given a fault in one of the transistors used in an inverter's functionality, the CCAB could just be reconfigured to use a different one.

IV. Solution

Given the significant configurability available within the PAnDA architecture, the authors decided to explore the approach that, should a fault be detected in a circuit, an evolutionary algorithm is used to try to find a new circuit configuration that works on the faulty substrate.

The presumption is that if the faulty structure is now treated as the environment, the evolutionary algorithm may be able to find a different way of configuring the same function in this new environment. The architecture of the PAnDA-EINS fabric may introduce some new possibilities within the realm of fault tolerance due to its great flexibility in terms of how individual logic functions can be implemented, on different levels.

V. Experiment

The solution proposed by the authors combines the PAnDA architecture with an evolutionary algorithm in order to overcome faults. A logic function is extrinsically evolved on a CCAB and then broken by introducing a "stuck at" fault in one of the transistors. The evolutionary algorithm is then used to again try to evolve the same function on the now faulty CCAB. This work is limited to investigation of single-faults, although investigating the resilience of the proposed solution to multiple faults is something to be looked into at a later stage.

A Multi-Objective (MO) GA (NSGA-II [14]) was used to evolve circuits on the substrate with a given functionality. This allowed the GA to optimize a number of different objectives at the same time without over-fitting on just one of them.

A SPICE [15] netlist was constructed for a CCAB, replacing the configuration inputs with voltage sources controlled by binary GA genes. There are 144 bits (32 for CT enabling and clamp configuration, 112 for width configurations) used to configure one CCAB. This size of the search space and the non-concavity of the function mean that the problem is not suitable for exhaustive search or a hill-climbing algorithm.

A. Tools

As mentioned previously, the NSGA-II algorithm was used to evolve circuit netlists from a netlist template. The configuration inputs of the reconfigurable structure, normally controlled by SRAM, were attached to simple voltage sources, which in turn had their voltages set by binary genes, essentially allowing the GA to make configuration bits either high (1V) or low (0V).

Ngspice [16] was used for simulating and measuring the circuits once they had been produced from the template. The transistor models used for this were the 25nm uniform transistor models from GSS [17].

B. Fitness Function

The fitness function designed for the experiment comprised several objectives, hence the choice of an MO algorithm [12] such as NSGA-II. Initial experiments used the number of correct truth-table outputs as the sole objective in order to try to evolve some basic functions and to see what configurations the evolutionary approach could find. Each truth table input combination would be applied to the circuit at least once, and the output measured. This method had a couple of serious drawbacks when it came to evolving logic functions, however: trying to evolve a 3-input NAND gate often resulted in a circuit that output a constant '1', which would score very high in the evaluation step and were difficult to move away from. In some cases, circuits would be evolved that contained a floating output during some input combinations. The floating output state would maintain whatever value the previous state had as the voltage would only drop slightly due to leakage current,

not enough to drain the charge within one input cycle, and so these circuits would often still score high. These issues were solved by both changing the input pattern so that the desired output pattern would change on every new state. A new objective was also added which used the RMS error measurement between what was desired at the output and what was seen. Since this objective also rewards circuits that have a low RMS error during the transitions, it is also related to the circuit's speed. Adding to the standalone propagation delay minimisation, this would reduce the power related objective's impact on the overall fitness, and therefore transitions were not analysed when calculating the RMS error, and this is addressed further in this section.

1) Truth Table: A voltage measurement was taken from the output at each stable state and then normalised to either 0V or 1V if it was below 0.05V or above 0.95V respectively. If it was measured to be in between these values, it was left as it was. The $1s$ and $0s$ were compared with a target truth table and the number of correct outputs were summed. The objective for the GA was then to maximise this number.

2) RMS Error: The Root Mean Squared (RMS) error objective compares the transient output of the circuit with the desired waveform. Any differences are squared (to bring them into the positive domain) and then averaged and square-rooted, to give the mean value. The RMS error objective was modified to analyze only the stable states, disregarding the transitions. This is because RMS error during transitions is related to speed and in order for us to control the evolutionary run, speed (or propagation delay) needed to be a separate objective. RMS error has the advantage over our Truth Table objective in that it is a continuous measurement rather than a discrete one, which gives a smoother fitness gradient that is useful for the evolutionary process.

3) Propagation Delay: The propagation delay of each transition of the output was found and then all of these added up and averaged. By minimising the average propagation delay, the algorithm would attempt to increase the speed of the circuit. The difficulty with this objective was how to manage transitions that didn't happen, for instance when the desired function wasn't being performed.

4) Dynamic Power: Power usage was measured as the dynamic power used over the course of the run. Power is mainly dissipated during transitions when the transistors switch states. By trying to minimise this, evolution is prevented from just maximising the widths of all the transistors, giving it a very fast transition time (low delay) but using a lot of power.

When evolving the initial function, which would then be broken, an objective was added to attempt to minimise the number of CTs enabled. This was to try to tidy up the circuits produced, getting rid of any non-functional CTs. When trying to fix a broken function, however, this objective was removed as it was found too restrictive.

The initial configuration for both experiments was to have all the CTs in the enabled state, with all the main functional PMOS CTs set to a width of 540nm and the NMOSs to 380nm. The CTs of the inverter that generates the inverted output of the circuit were fixed at 380nm for the PMOS and 240nm for the NMOS.

C. The NAND3 Benchmark

In the NAND3 experiment an optimised, 3-input NAND gate was evolved on the CCAB structure. It was then "broken" by fault injection, which took the form of forcing one of the functional transistors to always conduct. From this broken state, evolution was again used to try to evolve a working gate on the broken substrate.

1) Evolving from the Fault-Free Substrate: The first experiment consisted of evolving a 3-input NAND gate on a fault-free substrate. The initial conditions were set so that all of the 14 transistors in the CAB were enabled.

2) Evolving from the Broken Substrate: After having evolved an optimised 3-input NAND gate another experiment was carried out. One transistor, previously chosen by evolution to be enabled, was set to conduct constantly, affecting the behaviour of the evolved gate and crippling its functionality. Evolution was used once again to restore functionality to the design and to optimise it for power and propagation delay simultaneously.

D. The AOI21 Benchmark

In order to confirm that the same method can be applied to other gates that can be realised by a single CAB, a 3-input AND-OR-Invert function was evolved on a fault-free substrate, from the same initial conditions as the ones set for the NAND experiment. Similarly, an enabled transistor from the evolved design was set to insulate constantly, and evolution was used to restore functionality and also optimise it for power and propagation delay at the same time.

VI. Results

A. 3-input NAND Gate

Having set all of the transistors in a CCAB to be enabled, this gate was evolved. In the testbench, the input signals were used to perform a sweep of all the logic **zeros** and **ones** of a function: all input combinations that should yield a logic zero at the output were put between input combinations that should yield logic ones at the output, so that every transistion could be verified. With this setup, the output signal should then consist of a sequence of alternating logic zeros and ones, and any consecutive logic ones or zeros would represent a non-functional circuit/design.

The waveforms of the result of the evolutionary run can be seen on Fig. 3(a), and a complete schematic of the evolved configuration for a 3-input NAND can be viewed in Fig. 4, which represents the evolved circuit with the best performance in speed, chosen as the benchmark circuit. The NSGA-II algorithm was run for 100 generations of a population of 60 individuals, and the mutation rate set to 2%.

The next step was to simulate a fault in one of the transistors of the evolved design, and this was done as Fig. 5 demonstrates. Evolution was again used to create a new design in the presence of the faulty transistor, this time increasing

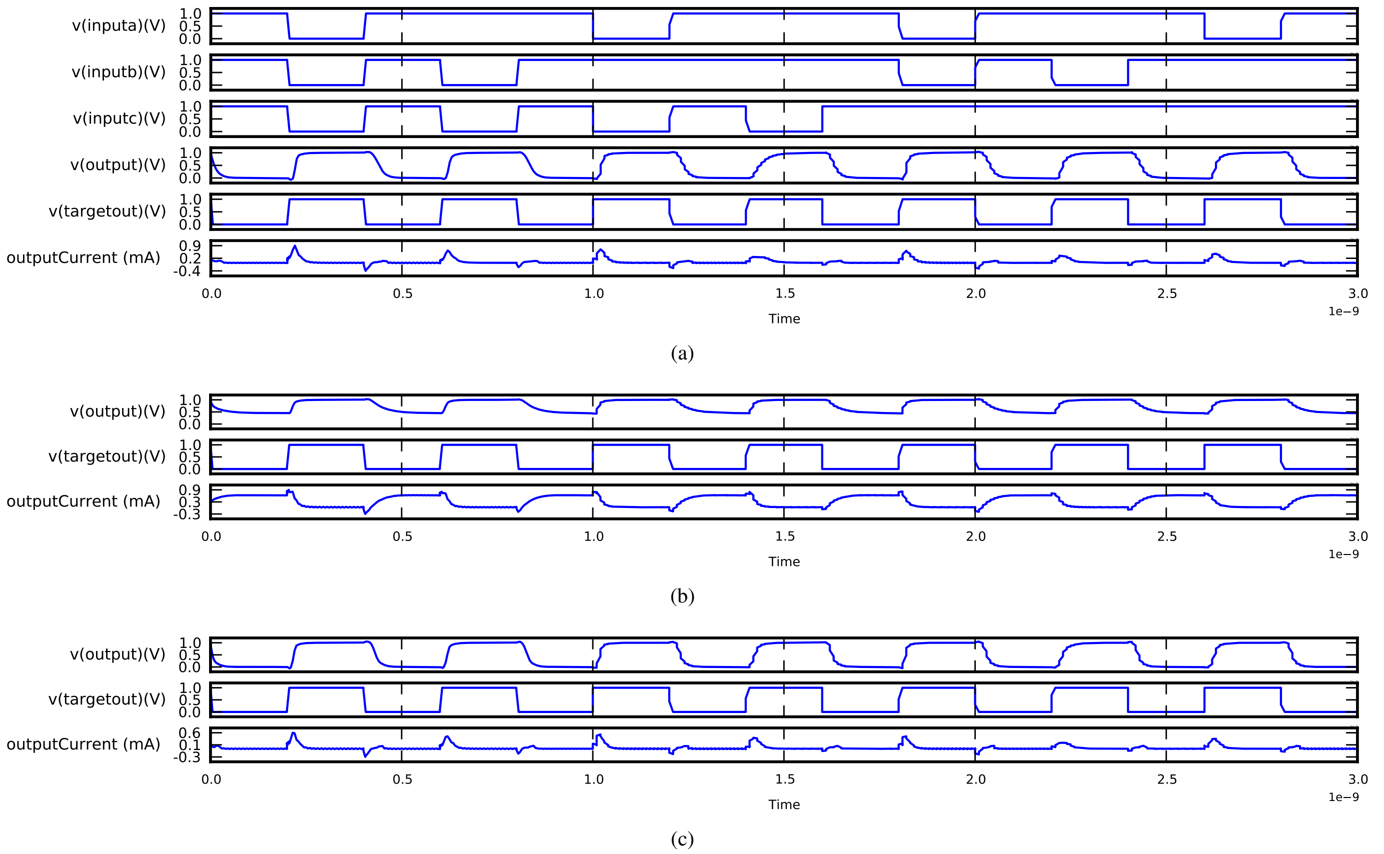


Fig. 3. A simulation of (a) the evolved 3-input NAND gate on the CCAB with the waveforms of the 3 input signals – A, B and C –, (b) the faulty 3-input NAND gate, and (c) the recovered 3-input NAND gate, showing the actual and target outputs and output current.

the NSGA-II population size to 116[4] and running it for 50 generations, with the same mutation rate as before.

Functionality was lost and power consumption increased dramatically, since the fault induced caused the output of the CCAB to be permanently connected to the power rail (Vdd). This can be verified by looking at the waveforms generated by this faulty circuit, and the rise in power consumption becomes evident with the generated current waveform, depicted in Fig. 3(b).

Fig. 3(c) shows the waveforms resulting from the simulation of the design evolved in the presence of the faulty transistor, which is shown in Fig. 6.

Table III shows the results obtained for the evolved gate, one evolved in the presence of the fault and also the recovered gates, individually optimised for speed and for power.

From this table it is possible to see that, as expected, the design that is optimised for speed implements larger sized transistors, whereas the one optimised for power uses smaller, and thus slower transistors. The average delay measured on the initially evolved NAND gate is higher than the one optimised on the faulty fabric perhaps because it wasn't evolved for a sufficient number of generations. It does, however, sit between the two evolved gates in terms of performance measurements.

[4]This number is derived from the number of cores on the cluster where the experiments were carried out

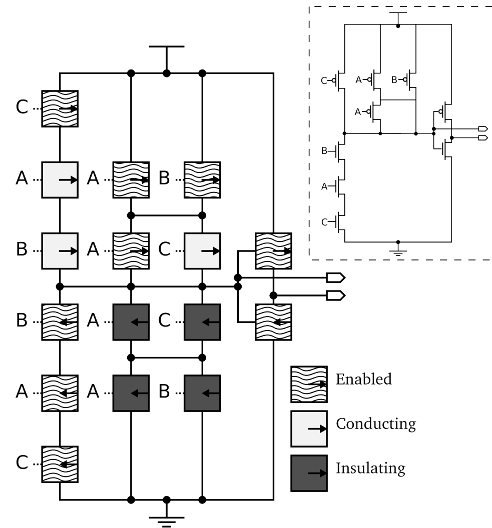


Fig. 4. The evolved design of a 3-input NAND gate, with a simplification of the effective circuit on the top right corner.

The average delay in the case of the faulty circuit is not shown since it is not relevant for our testbench, as it is an average measurement for a fully functional circuit. Nevertheless, the power measurement is illustrated so that the serious effects of

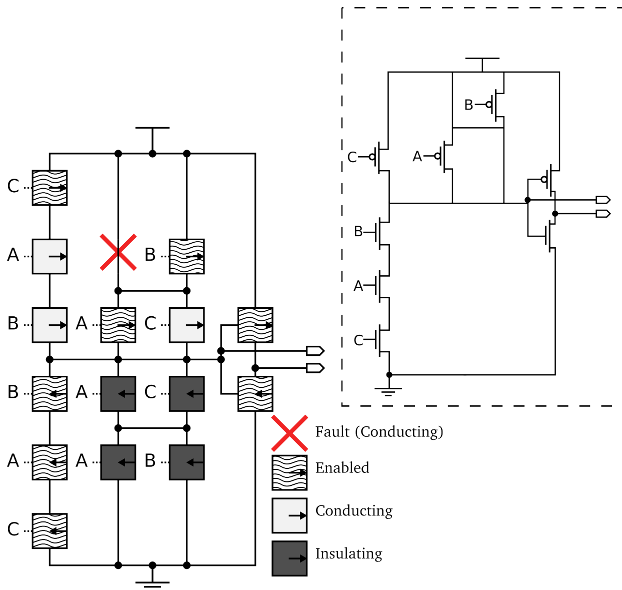


Fig. 5. The fault was induced in a previously enabled transistor on the evolved NAND3 design, making it conduct all the time and disabling the gate's functionality. A simplification of the effective circuit can be seen on the top right corner.

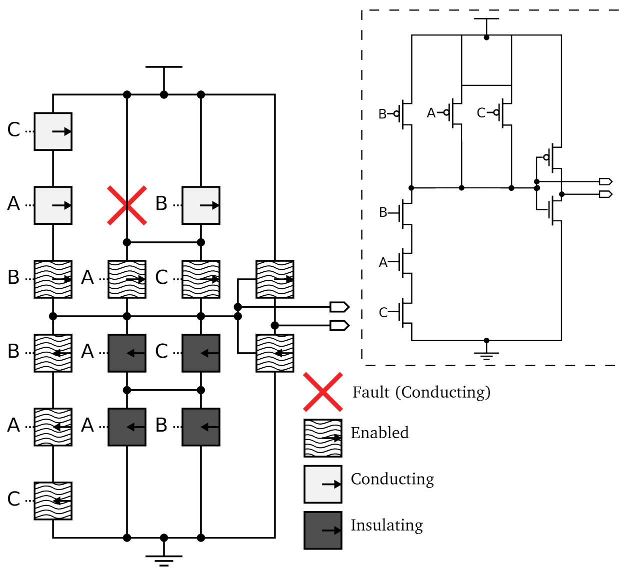


Fig. 6. The 3-input NAND evolved on the faulty fabric, with a representation of the effective circuit generated on the top right corner.

the induced fault can be observed.

The table proves that it is possible to restore both functionality and performance in power consumption and speed to a gate that experiences an extreme fault such as the one implemented in this experiment.

B. 3-input AOI21

To establish a performance benchmark, such as in the case of the NAND gate, a 3-input AOI21 (AND-OR-Invert) function was evolved on a fault-free fabric. An evolutionary run took place, with the same initial conditions as set for evolving

TABLE III
THIS TABLE SHOWS THE WIDTHS OF ALL THE TRANSISTORS OF THE INITIALLY EVOLVED 3-INPUT NAND GATE, THE FAULTY CIRCUIT, THE ONE BEST OPTIMISED FOR SPEED, AND THE ONE BEST OPTIMISED FOR POWER. THE LAST TWO ROWS DESCRIBE THEIR PERFORMANCE IN POWER CONSUMPTION AND PROPAGATION DELAY (AVERAGE, PER TRANSITION). TRANSISTOR NUMBERS RELATE TO THE CCAB SCHEMATIC FROM FIG. 2 AND ENABLED TRANSISTORS HAVE THEIR WIDTHS WRITTEN IN BOLD.

Transistor ID	Evolved NAND3	Induced Fault	Evolved for speed	Evolved for power
P1	**1020**	**1020**	1140	1020
P2	1020	1020	1140	1020
P3	1020	1020	**720**	**300**
P4	**1140**	1140	1140	1140
P5	**620**	**620**	**840**	**380**
P6	1140	1140	**880**	**340**
P7	**1000**	**1000**	1000	1020
N1	**1140**	**1140**	**1140**	**860**
N2	**1140**	**1140**	**980**	**400**
N3	**1020**	**1020**	**840**	**320**
N4	340	340	660	0
N5	520	520	460	160
N6	280	280	520	480
N7	800	800	600	400
Power(μW)	32.395	338.797	33.127	30.279
Avg Delay (ps)	50.764	N/A	45.451	61.780
Enabled Transistors	7	6	6	6

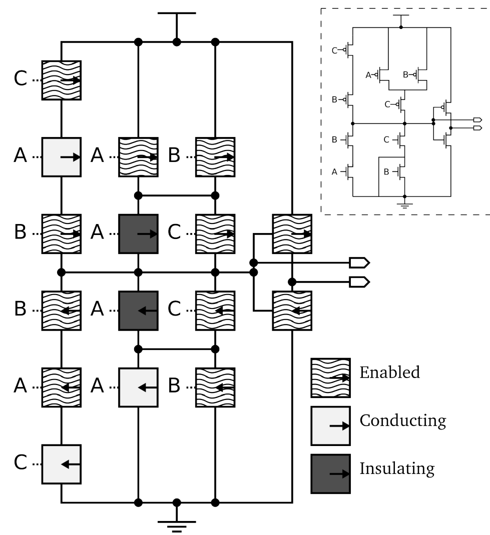


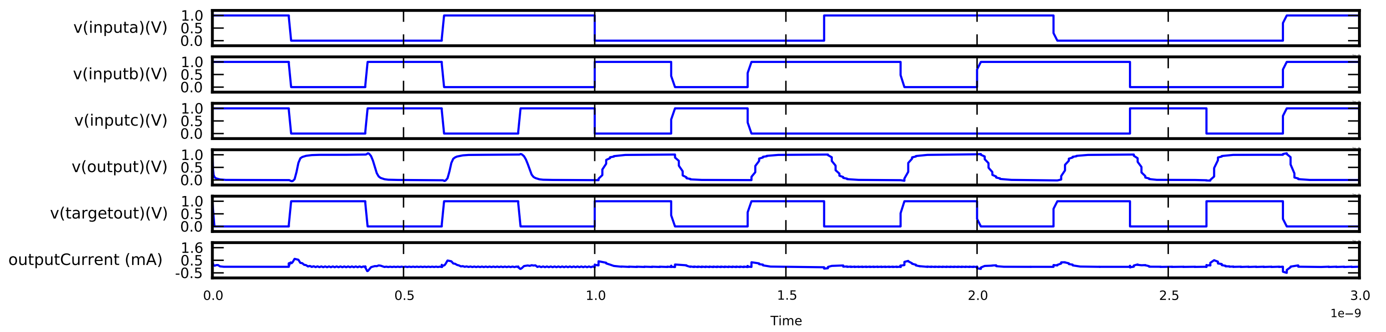
Fig. 8. The evolved design of a 3-input AOI21 gate, with a simplification of the effective circuit on the top right corner.

the NAND gate, and the NSGA-II algorithm was allowed to evolve circuits for 100 generations of 60 individuals, with a mutation rate set to 2%.

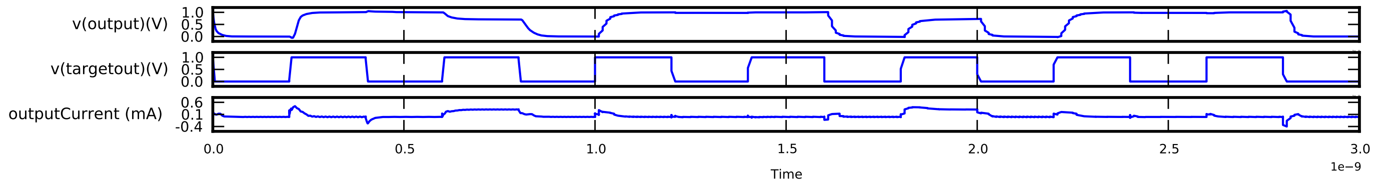
The circuit with the best performance in speed was chosen as the benchmark, and it is represented in Fig. 8. The waveforms generated by this circuit can be see in Fig. 7(a).

A similar fault to the one introduced in the case of the NAND was injected into one of the enabled transistors on the evolved design, in this case one of the enabled NMOS transistors, as shown in Fig. 9.

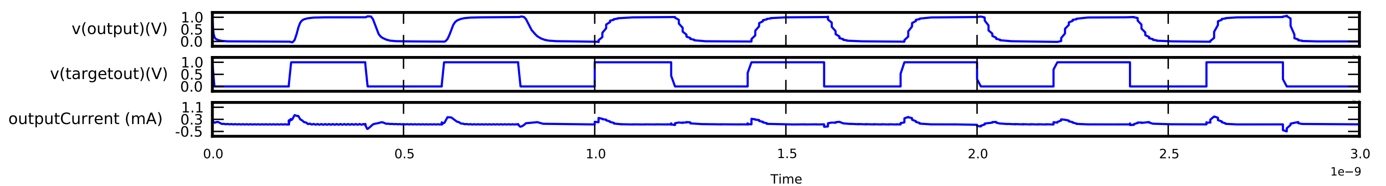
Again, the gate's functionality was lost, and evolution was



(a)



(b)



(c)

Fig. 7. A simulation of (a) the evolved 3-input AOI21 gate on the CCAB with the waveforms of the 3 input signals – A, B and C –, (b) the faulty 3-input AOI21 gate, and (c) the recovered 3-input AOI21 gate, showing the actual and target outputs and output current.

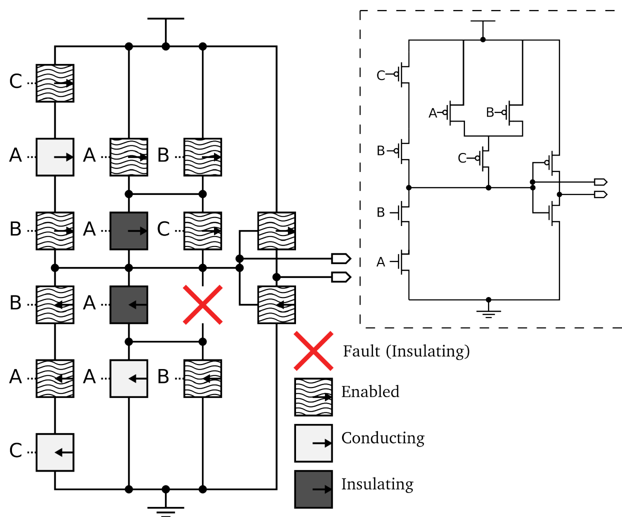


Fig. 9. The fault was induced in a previously enabled transistor on the evolved AOI21 design, making it insulate all the time and disabling the gate's functionality. A simplification of the effective circuit can be seen on the top right corner.

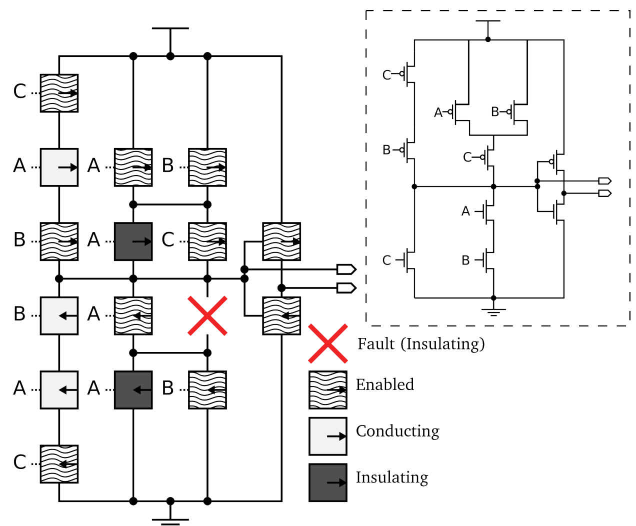


Fig. 10. The 3-input AOI21 evolved on the faulty fabric, with a representation of the effective circuit generated on the top right corner.

once again run for 50 generations of 116 individuals to try and restore both functionality and performance values to the gate. The evolved circuit with the best performance in speed is shown in Fig. 10.

A plot of the fault-induced and the repaired circuit's waveforms can be viewed in Figs. 7(b) and 7(c), respectively.

Having had functionality restored to the gate, a comparison of power consumption and speed is presented in Table IV.

TABLE IV
THIS TABLE SHOWS THE WIDTHS OF ALL THE TRANSISTORS OF THE INITIALLY EVOLVED 3-INPUT AOI21 GATE, THE FAULTY CIRCUIT, THE ONE BEST OPTIMISED FOR SPEED, AND THE ONE BEST OPTIMISED FOR POWER. THE LAST TWO ROWS DESCRIBE THEIR PERFORMANCE IN POWER CONSUMPTION AND PROPAGATION DELAY (AVERAGE, PER TRANSISTION). TRANSISTOR NUMBERS RELATE TO THE CCAB SCHEMATIC FROM FIG. 2 AND ENABLED TRANSISTORS HAVE THEIR WIDTHS WRITTEN IN BOLD.

Transistor ID	Evolved AOI21	Induced Fault	Evolved for speed	Evolved for power
P1	**880**	**880**	**980**	340
P2	740	740	800	660
P3	**660**	**660**	**660**	640
P4	**1140**	**1140**	**1140**	**580**
P5	700	700	880	0
P6	**1020**	**1020**	**880**	**600**
P7	**1140**	**1140**	**840**	**680**
N1	1140	1140	860	520
N2	**1140**	**1140**	**920**	**260**
N3	**1020**	**1020**	1000	200
N4	1140	1140	0	0
N5	0	0	**460**	**380**
N6	**880**	880	880	880
N7	**1000**	**1000**	**1000**	**540**
Power(μW)	38.934	56.446	43.655	36.792
Avg Delay (ps)	41.812	N/A	50.369	76.058
Enabled Transistors	9	8	8	6

Once again, both functionality and performance in speed and power consumption are restored to values that are very close to those prior to the fault occurence.

In the case of the design optimised for speed, represented in Fig. 10, one can state that the top left PMOS branch is a repetition of the top right branch, and therefore redundant. If evolution was allow to run for further generations, perhaps this redundancy could be removed and both power consumption and speed would be improved.

In any case, the circuit evolved for power on the faulty fabric actually identifies this branch as redundant and removes it from its design.

VII. CONCLUSION

From the results obtained with these experiments, where both functionality and performance values were restored to faulty circuits, it appears that the PAnDA fabric can be tolerant to faults that involve the enabling and disabling of its internal transistors, whether it be the transistors inside a CT or the entire CT itself. The latter is considered to be the worst-case scenario, and it was the one addressed in this paper.

The advantage of replacing the routing block and function decoder by configurable clamps connected to CTs is evident from the results. Without the configurable clamps, PAnDA-EINS CCABs only allow for a single configuration for each function, and so cannot be changed to tackle faults that occur in the fabric. PAnDA-ZWEI proves to be a flexible substrate that has the ability to restore a faulty design back to its fault-free working parameters.

The results presented in this paper contribute to confirm PAnDA-ZWEI as a flexible, potentially fault-tolerant and reconfigurable platform, suitable for the implementation of evolvable designs.

Intrinsic variability in circuit design is also an effect to be considered in further experiments. Some of its effects have already been modelled in this paper – such as the complete malfunction of a single CT – but other less obvious effects can occur, such as changes in the threshold voltage of transistors, potentially causing reductions in speed and power consumption performance. The solution proposed in this paper has the potential to overcome some of these effects in real-time on the chip, and it also has the potential to evolve designs that are tolerant to variability in the sense that they are resilient to changes in transistor properties within a certain range.

REFERENCES

[1] A. Asenov, "Random dopant induced threshold voltage lowering and fluctuations in sub 50 nm MOSFETs: a statistical 3D 'atomistic' simulation study," *Nanotechnology*, vol. 10, no. 2, pp. 153–158, Jun. 1999.

[2] M. A. Trefzer, J. A. Walker, and A. M. Tyrrell, "A Programmable Analogue and Digital Array for Bio-inspired Electronic Design Optimization at Nano-scale Silicon Technology Nodes," in *45th Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, 2011, pp. 1537–1541.

[3] J. Langeheine, S. Fölling, K. Meier, and J. Schemmel, "Towards a silicon primordial soup: A fast approach to hardware evolution with a VLSI transistor array," in *Proc. 3rd Int. Conf. on Evolvable Systems From Biology to Hardware (ICES2000)*, J. Miller, A. Thompson, P. Thomson, and T. C. Fogarty, Eds. Edinburgh, Scotland, UK: Springer Verlag, Apr. 2001, pp. 123–132.

[4] A. Stoica, D. Keymeulen, R. S. Zebulum, A. Thakoor, T. Daud, G. Klimeck, Y. Jin, R. Tawel, and V. Duong, "Evolution of Analog Circuits on Field Programmable Transistor Arrays," in *Proc. of the Second NASA/DOD Workshop on Evolvable Hardware*. Palo Alto, CA, USA: IEEE Computer Society Press, Jul. 2000, pp. 99–108.

[5] J. Langeheine, "Intrinsic Hardware Evolution on the Transistor Level," Ph.D. dissertation, 2005.

[6] M. A. Trefzer, "Evolution of Transistor Circuits," Ph.D. dissertation, 2006.

[7] M. Garvie, "Reliable electronics through artificial evolution," Ph.D. dissertation, University of Sussex, Brighton, 2005.

[8] G. W. Greenwood, D. Hunter, and E. Ramsden, "Fault Recovery in Linear Systems via Intrinsic Evolution," in *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*. Seattle, WA, USA: IEEE Press, Jun. 2004, pp. 115–122.

[9] R. O. Canham and A. M. Tyrrell, "Evolved Fault Tolerance in Evolvable Hardware ." in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 2, 2002, pp. 1267 –1271.

[10] D. Keymeulen, R. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmable transistor arrays," *IEEE Transactions on Reliability*, vol. 49, no. 3, pp. 305–316, 2000.

[11] J. a. Hilder, J. A. Walker, and A. M. Tyrrell, "Designing variability tolerant logic using evolutionary algorithms," *Ph. D. Research in Microelectronics and Electronics*, pp. 184–187, Jul. 2009.

[12] J. A. Walker, J. A. Hilder, D. Reid, A. Asenov, S. Roy, C. Millar, and A. M. Tyrrell, "The evolution of standard cell libraries for future technology nodes," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 235–256, Apr. 2011.

[13] J. A. Walker, M. A. Trefzer, and A. M. Tyrrell, "A Reconfigurable Architecture for Current and Future Challenges in Electronic Design and Technology," in *VAMM*, 2012.

[14] 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.

[15] P. Nenzi and H. Vogt, "Ngspice users manual," September 2010.

[16] "Ngspice circuit simulator." [Online]. Available: http://ngspice.sourceforge.net/

[17] "Gold Standard Simulations Ltd. :: Home." [Online]. Available: http://www.goldstandardsimulations.com/