Parity Waterfall Method

Jaroslav Borecký, Martin Kohlík, Hana Kubátová
Department of Digital Design, Faculty of Information Technology
Czech Technical University in Prague, Technická 9, Prague, Czech Republic
{borecjar, kohlimar, hana.kubatova}@fit.cvut.cz

***Abstract*—This paper proposes a method for improvement of the fault-coverage capabilities of Field Programmable Gate Array (FPGA) designs. It utilizes Concurrent Error Detection (CED) techniques and the basic architectures of actual modern FPGAs the Look-Up Table (LUT) with two outputs. Proposed Parity Waterfall method is based on a cascade (waterfall) of several waves of inner parity generating the final parity of outputs of the whole circuit. The utilization of the (mostly) unused output of a two-output LUT allows the proposed method to cover any single possible routing or LUT fault with a small area overhead. The method is experimentally evaluated using the standard set of IWLS2005 benchmarks and using our simulator/emulator. The experimental results of the proposed parity waterfall method are compared with a similar existing technique (duplication with comparison). These results show that the area overhead is smaller than the overhead of the duplication with comparison method for all of the tested circuits and 100% fault coverage is achieved.**

I. Introduction

Mission-critical systems with guaranteed level of safety and reliability parameters are used in many different applications (e.g. aviation, medicine, space missions, and railway applications, etc.) with different impacts to people and environment in a case of their failure.

These systems are often based on FPGA due to the universality of FPGA chips – the same chip can be used in many different applications only using alternations of its configurations. The development is cheaper and faster, because the price of the application based on an FPGA does not include costs of the development of the FPGA chip itself. Moreover, the possible reconfiguration without any hardware redesign can be used for either the recovery from a fault state or a less area overhead for pre-designed function changes (the whole functionality should not be implemented on a chip in the same time).

The main disadvantage of FPGAs is their sensitivity to many effects that can change their programmed function [1]. These changes are most unwelcome in systems, where financial losses, serious injuries or casualties can be caused because of a failure. The improvement of dependability parameters of the final design is required to minimize the impact of such effects.

A dependability of a system is the ability to avoid *service failures* (situations where the behaviour of the system deviates from the correct behaviour) that are more frequent and more severe than acceptable [2].

One of the most important technique allowing improvements of dependability is redundancy. This means that if one part of the system fails, there is an alternate functional part. However, redundancy can have a negative impact on a system performance, size, weight, power consumption, and others [3]. There are many redundancy techniques including hardware, information, time, software redundancy, etc. [3]. We focus on hardware redundancy made by replication in this paper.

A Fault Tolerant method designed for combinational circuits of newer FPGAs is proposed in this paper. This method covers all possible routing and LUT faults with a small area overhead. It is based on the architecture of FPGAs (the Virtex-5 family) and the technique of Totally-Self-Checking (TSC) [3], [4] circuits.

The proposed parity waterfall method is experimentally verified on the standard set of International Workshop on Logic Synthesis IWLS2005[5] benchmarks. The results demonstrate that the proposed parity waterfall methodology covers all possible routing and logic faults. The area overhead is smaller than the overhead of the duplication with comparison in 100% of the tested circuits.

The paper is organized as follows: Section II provides the theoretical background and introduces related models. The proposed parity waterfall method is described in Section III. The results are shown in Section IV and Section V concludes the paper.

II. Background

The proposed parity waterfall method utilizes the properties of the basic primitive of modern FPGA chips – the Look-Up Table (LUT) with two outputs. This primitive (LUT6_2)[6] can implement a function of 6-input logic or two functions of 5-input logic with shared inputs.

A. Self-checking Circuit

The main goal of the proposed parity waterfall method is to create a self-checking circuit – a circuit able to detect any single fault that may affect it. The self-checking circuit is mostly based on a generator of some kind of error detection code (i.e. parity generator, a copy of the original circuit, etc.). The outputs of this generator – the check bits – are connected (together with the outputs of the original circuit) to the checker that is able to determine whether the original circuit or the check bits generator is faulty or not. The checker must be able to detect faults inside itself to achieve a TSC system.

B. Common Hardware Redundancy Type

One well-known redundancy type is compared with the parity waterfall. This method – Duplication with Comparison (DwC) – is able to detect any single fault affecting the system, but it is unable to localize it. The duplication with comparison is not fully functional when a fault is detected.

A totally self-checking DwC is shown in Figure 1. A DwC contains two copies of the original circuit, two independent sets of inputs, two independent checkers (comparators), and two independent sets of outputs.

If any of these parts is present in a single copy only, it becomes a single point of failure and the system is not a DwC system anymore. A DwC system is able to detect any single fault, it may detect multiple faults, if there is at least one unaffected copy of the original circuit and comparator.

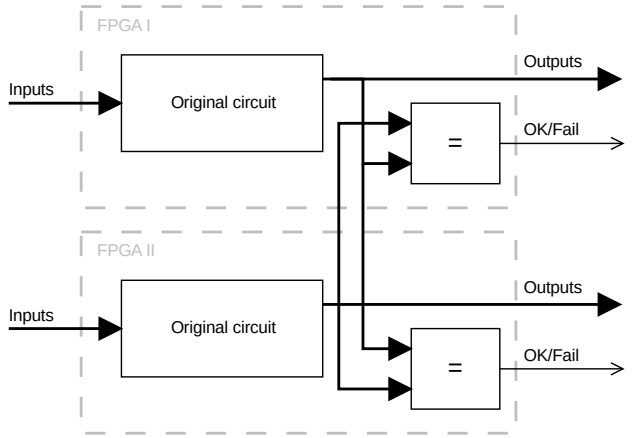

Fig. 1: Totally-Self-Checking duplex system.

C. Related Fault Detection Methods

The faults can affect content of Configurable Logic Blocks (CLBs) or the signal routing as described in [7].

The related fault detection methods are described below – they are sorted by the fault-detection locations they monitor.

1) Routing Faults: The method presented in [8] aims on the routing faults in FPGAs. It introduces two new CLB designs using a pre-computed parity value to detect error on any one of the input nets. However, the method uses off-line detection and it cannot be used in current FPGAs.

The slightly similar method is described in [9]. This selective Triple Modular Redundancy (TMR) (STMR) method is based on a modified circuit structure where the selected gates are tripled and the effectiveness of the method depends on the input signal probabilities. The area of STMR is smaller than TMR, but the fault coverage does not reach 100%.

2) LUT Content Faults: Most methods are based on the dual-output LUTs and use the unused parts of the LUTs to mask some faults in the LUTs by duplication [10], [11], [12] or use TMR method [13]. The method presented in [14] maximizes the fault-masking capabilities of a LUT using logic decomposition and restructuring.

Another method described in [15] proposing a new CLB architecture. The method applies the TMR method to a function generator that can generate any Boolean function with k-inputs.

The main disadvantage of these methods is a limited coverage of the faults (thus they do not cover all faults applicable to FPGA) and/or a requirement of a special architecture of CLB (thus they are not applicable on the current FPGAs). The method presented in [12] uses similar technique to the one presented in this paper, but it is based on duplication, thus it requires comparators to operate and it is not focused to achieve 100% fault coverage. Our proposed parity waterfall method combines several practices, it is able to cover all possible faults and uses the basic element of existing modern FPGAs.

III. Parity Waterfall Method

The main goal of the parity waterfall is to cover all possible routing and LUT content faults using a Totally-Self-Checking circuit with a parity generator with small area overhead at the same time. The parity waterfall block shown in Figure 2 performs the original function modified by the proposed Parity Waterfall (PWtf) method. The checkers are connected not only to the PWtf block outputs, but also to the PWtf block inputs, where the outputs of the previous block and/or primary inputs are checked.

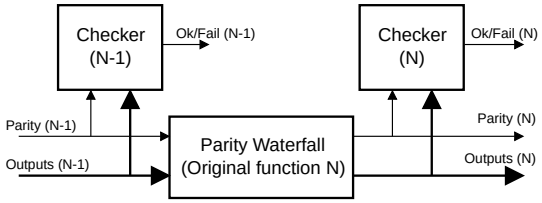

Fig. 2: Totally-Self-Checking circuit with Parity waterfall method.

The final parity of outputs of the whole circuit is calculated using a cascade (waterfall) of several waves of inner parities. The example of the cascade of parity waves is shown in Figure 3.

The first parity wave of the example is generated using the parity of the inputs of the circuit (A, B, C, and D) and the O5 (parity) outputs of the first level of the LUTs (two 2-input LUTs in the left part of Figure 3). The other waves are generated using the parity output of the previous wave and the O5 outputs of the last level of the LUTs. Each fault is propagated to the end of cascade and can be detected by the checker (parity predictor) connected to the PWtf block outputs.

The routing fault affecting any input of the LUT will change the value of the O5 output, therefore the change will be propagated as a single change to the parity wave at the next level. The same fault affects the O6 output as well, it will be distributed to other LUTs, where it will manifest as the routing fault of an input and is propagated as an another single change to the after-the-next parity wave. If the output of the circuit is affected by this fault as well and the parity output affected by two changes is correct, the parity predictor connected to the PWtf block outputs will detect it. If the output of the circuit is not affected by this fault, the checker of the next circuit will not detect it, but the output is correct and the fault is dormant (hidden). This detection scheme can fail, if the O6 output is connected to two (or the other even number) LUTs. More details about this situation and its solution will be presented in Section III-A.

The fault affecting the content of the LUT will change the O5 or O6 output (it cannot affect both output simultaneously), thus it can be treated as the routing fault on the inputs of the connected LUTs.

More detailed view to a single parity wave is shown in Figure 4. It is based on an original block composed from two LUTs (one LUT3 and one LUT2). The parts added by the proposed method are highlighted with the green color. The yellow colored blocks represent two identical logic functions. The bottom LUT6_2 presents the main idea of the proposed method. The method is based on LUT6_2 architecture, where

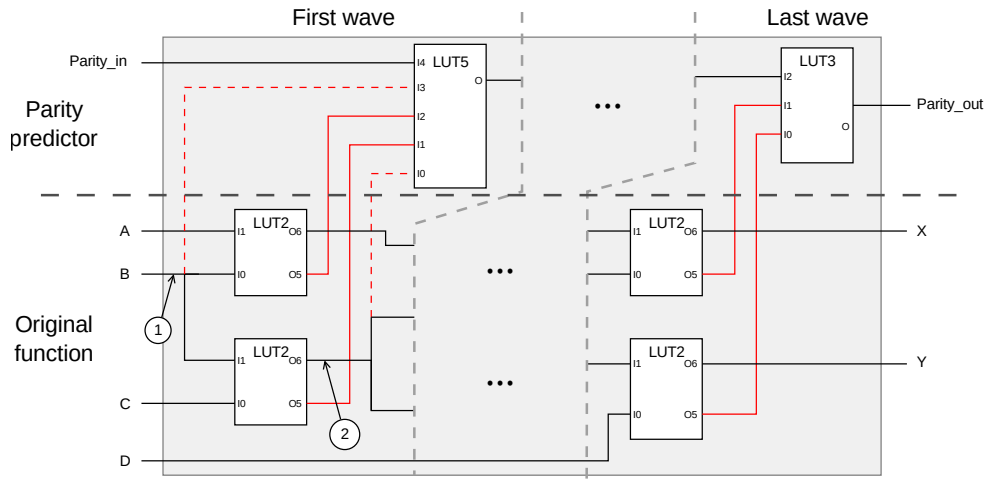

Fig. 3: Parity Waterfall wave cascades example.

two 5-input LUTs are multiplexed. If the original block contains LUT5 or smaller, LUT6_2 will be only configured partially, thus we can utilize the second LUT5 for our method.

The output X of the original function is generated using the output O6 of LUT3 (the output O6 of LUT6_2 would be used in a real FPGA). The output O5 of the same LUT is used to generate the inner parity for the parity of the first wave. The LUT5_bottom contains the function of original logic from LUT5_top, which is masked by the XOR function of LUT inputs (it performs a partial check of the inputs). This method is applied on each LUT of the original block. Then we add the last part composing all inner parities and the parity of inputs using the XOR function. This part not only composes inner parities together, but also unmasks the parity of outputs via the parity of inputs.

The following equations show the basic principle of parity unmasking for an example circuit shown in Figure 4.

$$\begin{aligned} parity\_of\_outputs &= (l\_func_{LUT2} \oplus inputs\_XOR_{LUT2}) \oplus (l\_func_{LUT3} \oplus inputs\_XOR_{LUT3}) \oplus parity\_of\_inputs \\ &= (l\_func_{LUT2} \oplus l\_func_{LUT3}) \oplus (inputs\_XOR_{LUT2} \oplus inputs\_XOR_{LUT3} \oplus parity\_of\_inputs) \\ &= (l\_func_{LUT2} \oplus l\_func_{LUT3}) \oplus 0 = X \oplus Y \end{aligned}$$

The *parity of outputs* output is created from the O5 outputs of both LUTs and *parity of inputs* input. If no error is present, *parity of outputs* is equal to $X \oplus Y$, but, when an error occurs, it is equal to $X \oplus Y \oplus 1$ (error at an input) or $X \oplus \bar{Y}$ (error in a logic function). The equation of the output checker is $X \oplus Y \oplus parity\_out$, thus checker output is equal to zero ($X \oplus Y \oplus X \oplus Y$), if no error is present. When an error occurs, the output is changed, therefore any single error can be detected.

The possible fault locations:
Inputs:

A, B, C, D, E – this fault can cause incorrect output and the same (undetectable) error at the output parity. But the output parity is changed to incorrect code word through unmasking part, because the input parity does not match parities from checkers of LUT inputs. The fault is detected by the checker connected to the block outputs.[1]

parity of inputs – the incorrect input parity is propagated to the output parity and can be detected by the checker, too.

I5 of LUT6_2 – this fault changes the output O6, which is now equal to O5 and the output parity is correct after unmasking, but it does not match the final outputs.

Outputs:

X, Y, parity of outputs – detected by the checker directly.

LUTs:

LUT5_top – this fault changes the output O6 and can be detected by the checker.[2]

LUT5_bottom – the inner parity is changed and produces the incorrect parity of outputs that can be detected by the checker.

LUT3 – the incorrect parity of outputs will be generated (and detected by the checker).

The method can be modified to perform an independent check of each parity wave output, but this modification is not used in this paper.

A. Routing Condition

The proposed Parity Waterfall method requires a specific routing condition. If the fanout of any signal is even (it is connected to the even number of other LUTs and/or outputs), it can cause incorrect output undetectable by the output parity. The proposed method avoids this situation by adding another branch to all signals with even fanout (represented by the dashed lines in the illustrative example).

[1] Only if the fanout of the input is odd – more details about even-fanout situation will be presented in Section III-A.

[2] Only if the fanout of the O6 output is odd – more details about even-fanout situation will be presented in Section III-A.

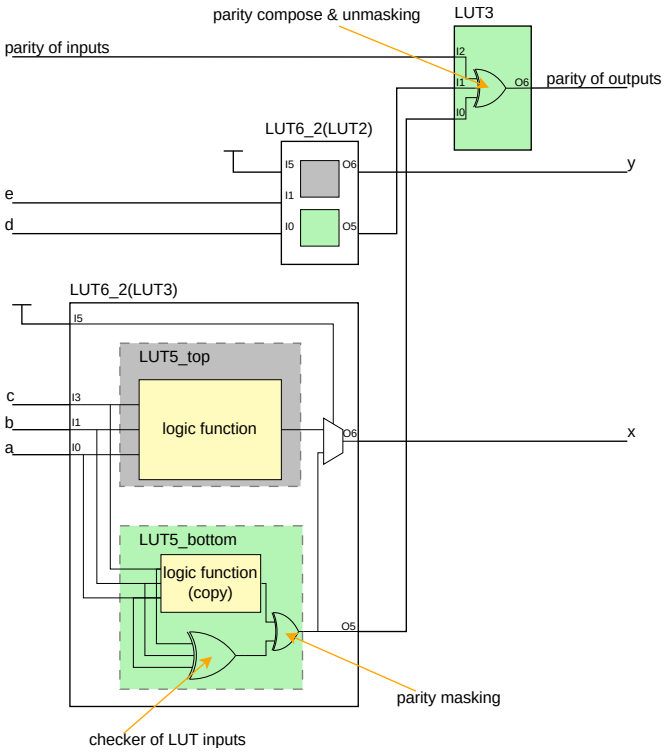

Fig. 4: Parity Waterfall method details.

If the split of the signal is performed correctly (see the signal marked as (1) in Figure 3), the fault can affect one branch of the signal, or all three. If a fault affects odd number of branches, it will be detected by the parity waterfall method. On the other hand, the signal containing the incorrect split (marked as (2)) contains a critical part (the black vertical part). The fault affecting this critical part affects two independent branches, thus it may not be detected.

The signal can be split several times, but all splits have to be performed correctly. The splits have to be corrected, when all components are placed and all signals are routed (after Place & Route step of the implementation). The automated tools performing these corrections are currently under development.

B. Method Application – Parity LUT Content Calculation

A logical function of LUT of a Virtex 5 is specified by 64-bit hexadecimal value stored in an INIT attribute. The upper half (bits 63:32) of the INIT value is used for the upper LUT5 and the lower half (bits 31:0) for the lower LUT5. The logic function of the O5 output corresponds only to the lower LUT5 value, but the O6 output can use both LUT5 values, depending on the I5 input value.

The parity waterfall method requires the function generating the O5 output of the LUT has to be configured as the result of XOR operation of the O6 function and the XOR of all inputs of the same LUT (see the green part of the LUT6_2(LUT3) in Figure 4).

The LUT6_2 primitive does not contain a XOR allowing connection between O6 and O5, but the INIT value – the content of the memory controlling the output of the LUT – of the lower LUT5 generating the O5 output can be calculated to effectively generate such XOR function, if the INIT value of the upper LUT5 generating the O6 output is known.

Parity LUT Content Calculation – Example

The following example shows, how the INIT value of LUT6_2 can be calculated. It is not necessary to know the original function, the waterfall method can perform the next steps using INIT value provided by any synthesis tool. The example is based on the INIT value of LUT4 (1) value provided by a synthesis tool.

$$LUT4\_INIT = 0145 \tag{1}$$

FPGA chips contain LUT6_2 primitives only, thus the INIT value must be extended to cover 5-input function. The extension of the 4-input function to a 5-input function (2) is a doubled copy of the original INIT value (1). This INIT value (2) guarantees that the value of the output will not depend on the fifth unused input.

The INIT value generating 5-input XOR function (3) does not depend on the original function. The 5-input XOR function depends on the fifth input (that was unused by the original function) – it allows the method to detect faults affecting this input.

Bitwise XOR applied on values (2) and (3) provides INIT value generating the O5 function of the LUT6_2 primitive (4).

LUT5:

$$LUT5\_INIT\_O6 = 01450145 \tag{2}$$

$$\oplus$$

$$INIT\_5\text{-}Input\_XOR = 96696996 \tag{3}$$

$$=$$

$$LUT5\_INIT\_O5 = 972C68D3 \tag{4}$$

The unused input can be also used to optimize the number of LUTs generating parity waves. An O5 output of a LUT of any of the previous levels can be connected to such input. The O6 function will not be affected, the O5 function will generate XOR including this input. If the O5 output is connected to the unused input, it does not need to be connected to the parity-wave logic and this logic may be reduced. The results presented in this paper are generated using this optimization.

The concatenation of the INIT values of the O6 (2) and O5 (4) functions forms the INIT value of the LUT6_2 primitive (5). The inputs I0-I3 are used to generate the original function and the inner parity, the input I4 can be used to optimize the number of parity waves, and the last input I5 must be connected to logic "1", the O5 function would be propagated to the O6 output otherwise.

$$LUT6\_2\_INIT = 972C68D301450145 \tag{5}$$

C. Preparation Flow

The standard IWLS2005 benchmarks are described in BLIF format and were used in our experiments. Process flow of the whole preparation and simulation is shown in Figure 5.

The process starts with creating an EDIF file. A source BLIF benchmark file is minimized and synthesized with the ABC[16] tool and it is executed with `dch;if -K 5;mfs`

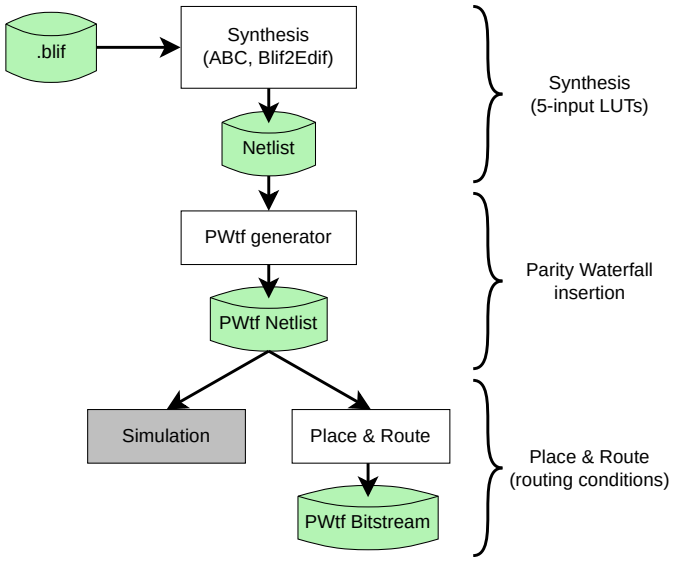

Fig. 5: Parity Waterfall preparation diagram.

parameters (the result will contain only LUTs with 5 or less inputs). Then is the final BLIF converted to EDIF by our Blif2Edif tool. These three steps can be skipped, if a circuit is described in EDIF format and contains only LUTs with 5 or less inputs.

This EDIF with original circuit is the input for our Parity Waterfall insertion tool. This tool creates a new EDIF containing all the necessary logic and signals that can be implemented in common tools or used for simulation in our software simulator.

The correction of the signal splits described in Section III-A would be performed after Place & Route step of the implementation. The automated tools performing these corrections are currently under development.

IV. Experimental Results

The proposed Parity Waterfall (PWtf) was tested on the set of standard IWLS2005 benchmarks. The benchmark source files has been synthesized and modified according to flow presented in Section III-C.

All results presented in this paper are results of the simulation based on post-synthesis EDIF file. The simulator is able to apply a single bit-flip fault to a LUT memory or a stuck-at fault at any input, output, or internal signal.

Two main parameters – the total size and the fault coverage – have been measured. The fault coverage measurement was performed for small benchmarks (containing up to 300 LUTs) only. Because the 100% fault coverage was achieved in all these small cases, the rest of the benchmarks has been used to measure total size only (The principles of this method do not depend on the size of circuits, the fault coverage of the bigger circuits will be 100%, too.) Therefore, the total size and the comparison between methods with the same fault coverage is the main focus of the results.

The results of the PWtf method and standard hardware redundancy method are shown in Table I. Original size means the size of the benchmark after synthesis and the values are the numbers of inputs and outputs (IOs), and used LUTs. The rests of the results are the sizes of the methods relative to the original size expressed as a percentage. The Overhead is only the redundant logic of the method, the PWtf and the DwC represent the sizes of the methods including the checkers. The number of IOs of the DwC method varies when the method is implemented in a single board or in two independent boards.

The Overhead values show that the PWtf method applied on the benchmarks achieved better results (less redundant logic) than the duplication itself.

The fault coverage of the new method is the same as the fault coverage of the DwC and the PWtf method has a lower overhead for all benchmarks.

Results show that the proposed PWtf method has the half size against the DwC method in most cases. Moreover, the number of IOs of the DwC method is doubled (or quadrupled) when compared with the original method, but the proposed PWtf method adds only three IOs (the parity of inputs, the parity of outputs, and the *OK/Fail* output of the checker).

V. Conclusions and Future Work

The Parity Waterfall (PWtf) method and the Duplication with Comparison (DwC) based on the parity waterfall allowing the detection of all faults affecting the circuit implemented in FPGA has been presented in this paper. The method has been applied on IWLS2005 sets of benchmarks to create self-checking circuits.

The main advantage of the method is the ability to create a self-checking circuit able to detect all faults affecting it without a fault simulation, thus it does not depend on a selected fault model.

The results indicate that the fault coverage of the presented method is able to detect all faults affecting LUTs and the routing logic. The overhead of the method has been compared with DwC method and the results indicate that the method has lower overhead than DwC in all benchmarks.

The parity waterfall method can be encapsulated into a scheme of duplication. This modification – Duplication with Parity Waterfall (DwPWtf) – will allow the detected faults to be localized, and the system to be fully operational during the fault repair (reconfiguration). This modification will allow us to compare the DwPWtf method to the Triple Modular Redundancy (TMR) system directly.

The presented method is valid only, if there are guaranteed specific routing conditions. These conditions are met, when there are no wires (signals) split into even numbers of branches (the fanout of any signal cannot be even). The method is able to fix these situations by adding an additional branch, thus the fanout of all signals is odd.

The routing of the FPGA must be modified to create correct (atomic) splits of all signals – splits not allowing a fault to affect only two independent branches keeping the third branch unaffected. This routing modification in FPGAs is not a part of this paper and will be processed in future work.

Acknowledgement

This research has been partially supported by the projects MSMT-32106/2015-1: LG15012 and GA16-05179S.

Benchmark	Original		Overhead	PWtf		DwC	
	(IOs)	(LUTs)	(%)	(IOs)	(%)	(IOs)	(%)
ac97 ctrl	132	3184	15.30	135	115.33	266[1]/458[2]	200.06
aes core	388	6783	13.70	391	113.71	778[1]/1294[2]	200.03
des area	304	1588	13.04	307	113.22	610[1]/866[2]	200.25
des perf	298	9946	18.97	301	118.98	598[1]/854[2]	200.02
ethernet	213	12642	15.08	216	115.09	428[1]/888[2]	200.02
i2c	33	277	12.27	36	112.64	68[1]/124[2]	200.72
mem ctrl	267	2362	8.43	270	108.47	536[1]/1144[2]	200.08
pci bridge32	369	5685	12.51	372	112.52	740[1]/1568[2]	200.04
pci conf cyc addr dec	64	32	15.62	67	121.88	130[1]/258[2]	212.50
pci spoci ctrl	38	280	10.00	41	110.36	78[1]/130[2]	200.71
sasc	28	172	20.35	31	121.51	58[1]/106[2]	204.65
simple spi	28	214	19.63	31	120.09	58[1]/106[2]	200.93
spi	92	1059	8.78	95	108.97	186[1]/366[2]	200.57
ss pcm	28	114	19.30	31	121.05	58[1]/94[2]	205.26
steppermotordrive	8	40	12.50	11	115.00	18[1]/34[2]	205.00
systemcaes	389	2256	8.91	392	108.95	780[1]/1296[2]	200.09
systemcdes	197	653	16.39	200	116.54	396[1]/656[2]	200.31
tv80	46	2297	9.53	49	109.58	94[1]/222[2]	200.09
usb funct	249	3899	8.54	252	108.59	500[1]/984[2]	200.21
usb phy	33	111	10.81	36	111.71	68[1]/140[2]	201.80
vga lcd	198	30854	11.62	201	111.63	398[1]/834[2]	200.01
wb conmax	2546	13244	12.61	2549	112.62	5094[1]/10758[2]	200.03
wb dma	432	1055	5.21	435	105.40	866[1]/1726[2]	200.57

[1] One board
[2] Two boards

TABLE I: IWLS2005 benchmarks: comparison of results of the Parity Waterfall and Duplication with Comparison methods.

REFERENCES

[1] Normand, E.: *"Single Event Upset at Ground Level, IEEE Transactions on Nuclear Science"*, vol. 43, 1996, pp. 2742–2750.

[2] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, *"Basic Concepts and Taxonomy of Dependable and Secure Computing"*, IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, January–March 2004.

[3] D. K. Pradhan, *"Fault-Tolerant Computer System Design"*, Prentice Hall PTR, Upper Saddle River, New Jersey 1996, ISBN 0-7923-7991-8.

[4] Kubalík, P., Kubátová, H., *"Dependable design technique for system-on-chip"*, Journal of Systems Architecture, Vol. 2008, no. 54, (2008) pp. 452–464. ISSN 1383-7621.

[5] Albrecht, C. IWLS 2005 Benchmarks. Technical report, June 2005.

[6] Xilinx, *Virtex-5 Libraries Guide for HDL Designs*, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/virtex5_hdl.pdf, April, 2015

[7] Bellato, M.; Bernardi, P.; Bortolato, D.; Candelori, A.; Ceschia, M.; Paccagnella, A.; Rebaudengo, M.; Reorda, M.S.; Violante, M.; Zambolin, P., *"Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA"*, Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, vol.1, pp.584,589 Vol.1, 16-20 Feb. 2004, ISBN 0-7695-2085-5.

[8] Syam Sundar Reddy, E.; Vikram Chandrasekhar; Sashikanth, M.; Kamakoti, V.; Vijaykrishnan, N., *"Detecting SEU-caused routing errors in SRAM-based FPGAs"*, VLSI Design, 2005. 18th International Conference on, pp.736,741, 3-7 Jan. 2005, ISBN 0-7695-2264-5.

[9] Samudrala, P.; Ramos, J.; Katkoori, S. Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *Nuclear Science, IEEE Transactions on*, volume 51, no. 5, Oct 2004: pp. 2957–2969, ISSN 0018-9499.

[10] Ju-Yueh Lee; Yu Hu; Majumdar, R.; Lei He; Minming Li, *"Fault-tolerant resynthesis with dual-output LUTs"*, Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific, pp.325,330, 18-21 Jan. 2010, ISBN 978-1-4244-5765-6.

[11] Leveugle, R.; Ben Jrad, M., *"On improving at no cost the quality of products built with SRAM-based FPGAs"*, Quality Electronic Design (ASQED), 2013 5th Asia Symposium on, pp.295,301, 26-28 Aug. 2013, ISBN 978-1-4799-1312-1.

[12] Ben Jrad, M.; Leveugle, R., *"Automated design flow for no-cost configuration error detection in sram-based FPGAs"*, Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on, pp.1,6, 9-11 Dec. 2013, ISBN 978-1-4799-2078-5.

[13] Kyriakoulakos, K.; Pnevmatikatos, D., *"A novel SRAM-based FPGA architecture for efficient TMR fault tolerance support"*, Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on, pp.193,198, Aug. 31 2009-Sept. 2 2009

[14] Das, A.; Venkataraman, S.; Kumar, A., *"Improving autonomous soft-error tolerance of FPGA through LUT configuration bit manipulation"*, Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on, pp.1,8, 2-4 Sept. 2013.

[15] Rohani, A.; Zarandi, H.R., *"A New CLB Architecture for Tolerating SEU in SRAM-Based FPGAs"*, Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on, pp.83,88, 9-11 Dec. 2009

[16] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. 2000, http://www.eecs.berkeley.edu/˜alanmi/abc/.