ROST-C: **R**eliability driven **O**ptimisation and **S**ynthesis **T**echniques for **C**ombinational Circuits

Satish Grandhi†, David McCarthy†, Christian Spagnol†, Emanuel Popovici†, Sorin Cotofana*
† Department of Electrical and Electronic Engineering, University College Cork, Cork, Ireland
* Faculty of EE, Mathematics and CS, Delft University of Technology, Delft, The Netherlands
sagrand@ue.ucc.ie, david.mccarthy@ue.ucc.ie, christian.spagnol@ue.ucc.ie, e.popovici@ucc.ie, s.d.cotofana@tudelft.nl

***Abstract*—Traditional logic synthesis methodologies are driven by timing, power, and area constraints. However, due to aggressive technology shrinking and lower power requirements, circuit reliability is fast turning out to be yet another major constraint in the VLSI design flow. Soft errors, which traditionally affected only the memories, are now also resulting in logic circuit reliability degradation. In this paper, we present a systematic and integrated methodology to address and improve the combinational circuit reliability measured in terms of Soft Error Rate (SER). The proposed SER reduction framework makes use of rewriting based logic optimisation technique which employs local transformations. The main idea behind our proposal is to replace parts of the circuit with functionally equivalent but more reliable counterparts chosen from a pre-computed subset of Negation-Permutation-Negation (NPN) classes of 4-variable functions. Cut enumeration and Boolean matching driven by reliability aware optimisation algorithm are used to identify best possible replacement candidates. Our experiments on a set of MCNC benchmark circuits and 8051 micro controller functional units indicate that the proposed framework can achieve up to 75% reduction of output error probability. On average, about 14% SER reduction is obtained at the expense of very low area overhead of 6.57% that results in 13.52% higher power consumption.**

Keywords*—And-Invert Graphs (AIG), ABC, Optimisation, Rewriting, Cut Enumeration, Boolean Matching, NPN Equivalence, Reliability, Soft Errors.

I. Introduction

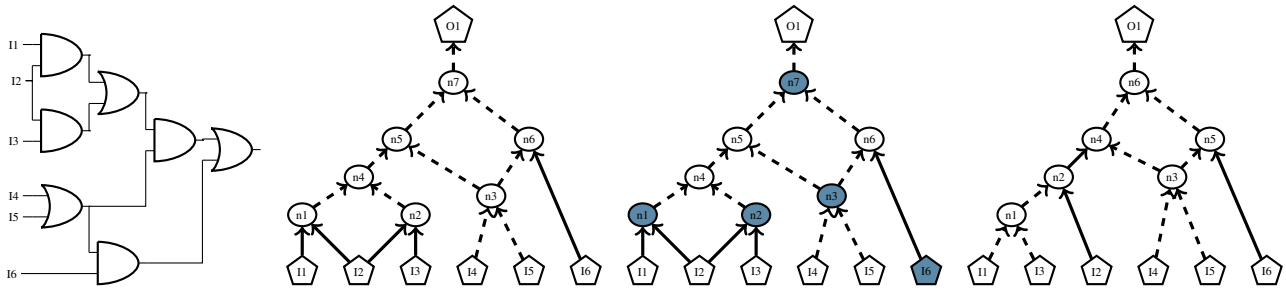
In the era of deep submicron CMOS technology, spatial and temporal variability is resulting in less predictable device behavior [2]. Stochastic logic delay variation on par with the nominal delay can occur due to the local (or intra-die or within-die) variations in transistor V_{th} [13]. Further, circuits operated in the near or sub-threshold region in order to achieve substantial power savings results in an increased amount of output Soft Error Rate (SER). In view of the combined effect of variability and aging mechanisms, state of the art CMOS gates can be regarded as being highly unreliable. To overcome reliability related concerns, a variety of system/circuit level techniques like Dynamic Reliability Management (DRM), and inexact computing [14], [15], [12] have already been proposed. These techniques are at higher abstract level and don't leverage the significant gains that can be achieved by employing intelligent graph modification/altering techniques. Reliability driven logic synthesis, is one area that has not received much attention but is gaining lot of importance in the last few years. In [7], the circuit output Soft Error Rate (SER) is reduced through localized circuit restructuring by taking advantage of don't care based re-synthesis and local rewriting. In [16], a technique to improve the circuit robustness to soft errors based on Redundancy Addition and Removal (RAR) by eliminating gates with large contribution to the overall SER is proposed. Efficient algorithms for synthesizing approximate circuits for concurrent masking of logical and timing errors were employed in [4]. ATPG-based rewiring method to generate functionally-equivalent yet structurally-different implementations to reduce the SER was developed in [1].

This paper introduces a novel reliability driven circuit optimisation and synthesis flow based on the standard cut-rewriting technique to improve the reliability of circuits built out of unreliable gates. An And-Invert Graph circuit (AIG) representation is utilized to represent the logic circuits. Rewriting of 4-input cuts by using a subset of Negation-Permutation-Negation (NPN) equivalent logic configurations is employed to improve circuit reliability. Key differences in the cut enumeration and Boolean matching techniques when constraint of optimisation is changed from area to circuit reliability are discussed in detail. While optimizing for area, the Boolean matcher strives to pick the best possible cut with least number of nodes. In order to improve circuit reliability, extra redundant nodes can actually help in masking gate errors thereby reducing output error. ABC [3], a logic synthesis and verification tool which performs scalable logic optimisation based has been used to accommodate all our algorithms. The tool working procedure is explained in detail with the help of a case study on the MCNC benchmark circuit $'cm162a'$. The proposed reliability aware logic synthesis methodology that applies the transformation rules in a guided fashion on complex combinational circuits is detailed out. Evaluation of the tool on a set of standard MCNC benchmark circuits and also the standard arithmetic circuits is performed. Results show an average reduction in output error of upto 14% while a peak improvement of upto 75.5% is observed. Performance analysis using Synopys Design Compiler shows that very low area overhead of 6.57% that results in 13.52% higher power consumption is the extra cost incurred.

This paper is organized as follows. Section II provides basic gate level synthesis background information and the terminology utilized in this paper. Section III introduces the reliability driven circuit optimisation algorithm as well as the associated circuit error probability computation methodology. Section IV illustrates the potential practical implications of our approach by means of a case study and evaluations performed on standard MCNC benchmark circuits. Section V concludes the paper and provides directions for future work.

II. AND Invert Graphs & Basic Terminology

And Invert Graphs (AIGs) are common circuit representation means where each graph node corresponds to a two-input AND gate and the edges, representing the node interconnections, optionally having an inverter. AIGs were proposed originally by Kuehlmann [8] for compressing circuits to speed up verification operations. An example circuit in both schematic and AIG form is presented in Fig. 1(a) and Fig. 1(b) respectively. Circles represent AND nodes, solid lines represent direct gate connections and dashed lines represent connections through an inverter. A key AIG feature is that they are always structurally hashed meaning that no two identical nodes (same



(a) Reference Circuit *(b)* AND Invert representation *(c)* '4' cut on node n7 *(d)* Boolean Equivalent

Fig. 1. And Inverter representation of Combinational Circuit

parents and same inversion) are permitted. Next, we define some of the common terminology used throughout the paper.

– **Cut** : A cut of a node N in a network is a set C of nodes such that any path from Primary Inputs (PI) to N passes through one of the nodes from the set [9]. Obviously, node N itself forms a trivial cut. The size of a cut refers to the number of leaves, rather than to the number of interior nodes. Cuts of up to a specific size k are called k-cuts or k- feasible cuts. An example is depicted in Fig. 1(c). The node set {I6, n1, n2, n3} defines a 4-cut C on the node n7. C is a 4-cut since it has four leaves: n1, n2, n3, and I6. Clearly, every path from a PI to n7 passes through at least one of them. Nodes n4, n5, and n6 are the internal nodes of the cut.

– **NPN equivalence** : Two Boolean functions, F and G, are NPN-equivalent , i.e., belong to the same NPN equivalence class, if F can be transformed into G through negation of inputs (N), permutation of inputs (P), and negation of the output (N) [6]. For example, the sub circuit comprising nodes {n1, n2, n4} in Fig. 1(b) and Fig. 1(d), evaluating the function G and F, are NPN-equivalent. G can be transformed to F by employing the following procedure: (1) negate I3 and I1, (2) swap the position of I3 and I2, and (3) negate the output node.

– **Boolean matching** : It is a technique widely used in technology mapping [10]. It is the process of replacing a sub graph with another functionally equivalent sub-graph. It is normally done by calculating the canonical form representation of functions by employing simple methods like negation and swapping of the nodes.

III. Reliability Driven Logic Synthesis

Logic synthesis is the process of transforming higher level functional representation into gate level representation. Typically, there are two main approaches to logic synthesis. The first is rule based synthesis which relies on searching the graph for specific substructures and replacing these with alternatives [5]. The second approach to synthesis is cut based rewriting [11]. In this section, we introduce an AIG 4-input cuts based rewriting algorithm meant to be utilized in reliability aware logic synthesis. We note that a key element in developing such an efficient optimisation and synthesis tool is the availability of accurate reliability information as well as efficient/fast algorithms for computing the reliability of logic functions representing partial solutions during the optimisation process. In view of this remark, we first describe reliability estimation methodology employed within our tool. Later, we describe the process of logic optimisation.

A. Reliability Estimation Methodology

Logic circuit reliability analysis attempts to evaluate the impact that the gate errors could have on the circuit Primary Outputs (PO) correctness. Traditional approach to reliability analysis begins with elementary SPICE simulations which is practically not feasible due to a prohibitive computation time and excessive resource requirements. Fig. 2 graphically presents the unreliable gate model we employed to inject errors onto the gate outputs with a pre-defined gate error probability. We modeled an unreliable AND gate as an ideal (error free) AND gate followed by a faulty XOR that determines the stochastic error behavior by toggling the gate output with a pre-defined probability. This model moves the entire error statistic on the output and so it implicitly assumes a symmetrical error behavior in relation to the inputs.

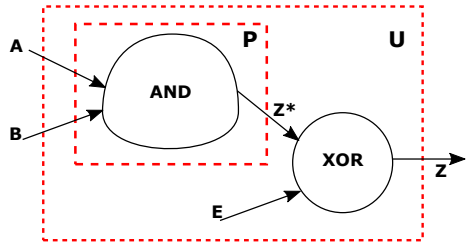


Fig. 2. Gate Error Model.

The problem of computing reliability information for combinational circuits is investigated by employing two different approaches. The first approach consist of simulating the combinatorial circuit under test and injecting faults. One of the most important issues related to the gate level simulation technique is the availability of significantly random input vectors. The traditional random function $rand$ from the 'C' library provides random numbers with a small period of length 2^{32} - 1. Since the number of required simulation steps runs into millions, there is a high possibility that patterns are repeated and highly correlated with each other. To overcome this problem, we used a pseudo random number generator called Mersenne twister. The name has derived from the concept of Mersenne prime. In mathematics, a Mersenne prime is a prime number of the form M_n = 2^n - 1 and has a very long period of 2^{19937} - 1. Thus, it helps us in overcoming all the limitations. The second method called Conditional Probabilistic Error Propagation (CPEP) builds a probabilistic error model of the combinatorial circuit and calculates the reliability of the circuit based on a mathematical analysis of the model. This is quite generic in nature and can be applied for any error scenario and for any logic gate.

CPEP methodology is used in intermediate steps where it is required to compare two configurations to make a fast decision of choosing the better configuration. Exact reliability numbers are not required in this stage which enables us do with slightly less accuracy. The simulation based methodology is very accurate but can be time consuming specially for large circuits. This is used in final steps which require accurate reliability numbers before signoff. Experimental results obtained with the proposed CPEP framework are within 2% average error and up to 1000X faster when compared to Monte Carlo simulations. Complete details of these techniques have been previously presented in [].[1]

B. Cut Based AIG Rewriting

This section presents an AIG 4-input cut based rewriting algorithm which targets the optimisation of circuit reliability rather than area, delay or power. Rewriting is a common logic optimisation approach which relies on local transformations. Most commercially available logic synthesis tools include a rewriting engine that may operate multiple times on the same netlist during the optimisation process. For traditional constraints like area/or delay, the goal is to reduce the number of nodes and the circuit, respectively. But, when circuit reliability is the optimisation constraint, the entire optimisation algorithm and its output changes drastically.

In cut based rewriting, the first step is to pre-compute the best circuits for a subset of NPN classes of 4-cuts from all possible such functions. Forest (list of all useful cuts) construction is done once and saved to a file, and when rewriting a circuit, the forest is loaded from a file rather than being computed each and every time. The basic forest constructed in this work consists of 2004 nodes implementing 106 NPN function classes. We note that ABC's original area based rewriting uses a forest of 1785 nodes implementing 139 function classes. This is because in the ABC's default rewriting algorithm targeting area optimisation, unit XOR nodes are also used in the forest but these are excluded from the reliability aware rewriting algorithm since XOR node can result in huge output error.

The second step is cut-enumeration and Boolean matching. Cuts that are rooted at a certain node are identified and functionally equivalent alternative cuts from the precomputed forest are evaluated as replacements for that node. For each cut, the nodes that are computing this function and no others can be removed. New nodes that aren't duplicates of existing nodes must be added. The cut implementation is then scored by the number of nodes it removes less the number it must add. Each node in the forest exists as a function itself and also as a fan-in for further nodes. Auxiliary tables and linked lists are maintained grouping each node in the forest into NPN function classes. The conditions for a node being tried to be added to the forest is that it is either less than 2 levels deep or implements an more complex NPN function class deemed practical, and that it is better than each existing function implementing that class in its output error probability.

The decomposition module within ABC encompasses all the coding details that handle network modifications during rewriting. A decomposition includes the new nodes that must be added or reused (found in the network by strashing). Nodes in the original cut are marked for removal if they are not needed elsewhere. In area based rewriting, a decomposition must be formed for every implementation of every cut as building the decomposition is how area change is calculated. All these decompositions are still built when rewriting for reliability so as to retain the same, known correct, code structure, even though the decomposition could be delayed and one calculated per cut, since the reliability information is available in the forest. For each alternative implementation, a decomposition (possible alteration) of the network based on substituting that implementation is formed.

[1]This reference is intentionally left blank to facilitate blind revision

Algorithm 1 Rewrite for Reliability

```
INPUT : P_ntk, NODE_count, PI_sp , G_err
OUTPUT: PO_err
for all nodes I= 1 to NODE_count do
  for Node N in AIG do
    Get cuts based at N
    for 4-input cut C based at N do
      Get truth-table F of N in terms of C
      for Possible graph S of function F do
        Make decomposition D corresponding to cut C
        Remove original nodes from D
        Add nodes of S to D
        if Level > MaxLevel then
          Go to Next S
        end if
        Compute savings as nodes that can be dropped from
        network with D
        Compute cost of adding new nodes.
        Error = Error(S)
        if Error < BestError then
          BestS = S
          BestD = D
        end if
      end for
    end for
    Apply decomposition BestD to the AIG
  end for
end for
```

C. The CAD Algorithm

The main algorithm implemented in this work is called "Rewrite for Reliability" or $'rwrel'$ for its command in ABC. This algorithm is based on the standard cut-rewriting algorithm from ABC, adapted so that the goal is reliability instead of area. Modified versions of the data structures were created to include node error probability values. The key functions of the rewriting process were then rewritten to use reliability as a goal. All utility functions from the rewriting module were duplicated, just changing them to work with the expanded structures instead of the originals. The cut enumeration and decomposition modules are self contained and are used directly. Eq. 1 quantifies the 'percent decrease in error' goal function that is used in guiding the optimisation algorithm to select the better cut.

$$R_{Metric} = \frac{P_\epsilon(Old\_Cut) - P_\epsilon(New\_Cut)}{P_\epsilon(Old\_Cut)} \quad (1)$$

A pseudo-code of this algorithm is presented in Alg. 1. The aim is to find a sequence of transformations leading to an AIG network that minimize the cost function. We rely on a heuristic approach to find an acceptable solution, i.e., an AIG providing a higher reliability than the original circuit. The strategy chosen is to figure out all the functionality equivalent cuts for a given node and choose the best

among them. Given a combinational circuit implementing the Boolean function f and its AIG network, the algorithm, starting from PI's, traverses through the graph to see if any of the transformations are applicable on the given node. For every possible transformation, the new circuit reliability value is computed. The configuration that yields the highest circuit reliability improvement is chosen and the new topology is generated. This process is continued on every graph node until we reach the primary outputs where no more transformations are applicable.

Two versions of the algorithm were developed in this work. In both of them, the rewrite for a given cut is selected for the least error, based on the reliability information available in the forest. Selecting which cut to use for node rewriting is the difference between the two algorithm versions. The first version selects the least error for each cut, and then selects the least error cut for the node. The second version instead selects the most improved cut in terms of percentage improvement in reliability, for each node. Initially the first version of the algorithm was implemented but while the obtained results were good, better performance was expected. One possibility considered was that the absolute error goal function favoured situations in which the optimisation process ended up in local minimum. Choosing the cut with best reliability naturally favours already optimized cuts where little progress can be made, where it might be more favourable to select a slightly unreliable cut and improve it to have better reliability. Thus, the second algorithm version makes use of reliability improvement as a metric instead of the absolute error. This metric change resulted in about three times better improvement and in view of this all the results presented in the next section are based on the second algorithm.

IV. Experimental Results

In this section, the potential practical implications of the proposed reliability aware synthesis tool is evaluated. In this view, Alg. 1 was implemented and integrated in ABC synthesis tool as a command $'rwrel'$. Another command $'printrel'$ is developed to display the node reliability statistics in the network. The total savings achieved in terms of circuit reliability is studied by performing a set of experiments using MCNC benchmark circuits [17] with more than 3000 node AIGs after structural hashing. A unit gate error probability of 10^{-2} is assumed in all our simulations. The networks were loaded in Berkeley Logic Interchange Format (BLIF) format and strashed to AIGs. Functional equivalence was extensively verified during development using the ABC equivalence check command $'cec'$. We use internal technology mapping algorithms to map the AIG network to a gate level Verilog netlist using the TSMC 65 GP CMOS standard cell library. Synopsys Design Compiler is employed for power, timing and area estimation on the two netlists. Also the two netlists are verified for equivalence via Synopsys Formality. After we established the algorithm correctness and performance, its effect on the implementation of 8051 microcontroller basic blocks like adder, multiplier, and divider were also tested. In this section, we first present a case study highlighting the important differences in the way the Boolean matcher finds alternative cuts when reliability is the constraint. Then, we discuss the results obtained for various MCNC benchmark circuits and 8051 functional units.

A. CM162a – A Case Study

The proposed reliability aware synthesis algorithm is applied on the MCNC benchmark circuit $'cm162a'$, which implements a 14-input 5-output logic function. The current case study considers the cone comprising of all the gates driving the output pin 'P'. Fig. 3(a) represents the default AIG configuration. The total number of nodes in the default AIG representation of the circuit is 14. With a unit gate error probability of 10^{-2}, the output node 'P' has an error probability Err_{org} = 0.071750. We now discuss three specific functionally equivalent logic configurations generated by our tool. The idea is to emphasise on how the Boolean matcher works when the constraint is set to output error optimisation instead of area.

Fig. 3(b) depicts the first sub-case : Moving the PI's closer to the PO's wherever possible. It is clear that the PI node 'F' has been pushed closer to the PO. This helped in reducing the overall output error on the output node to Err_{rwrel1} = 0.068002. Fig. 3(c) depicts the second sub-case: insert redundant nodes in an intelligent fashion. Extra nodes if added pragmatically can result in significant error masking effect that reduce the overall error on the output nodes considerably. A redundant node $'n6'$ has been appended onto the logic, thus total node count is increased to 15 and reduces the overall output error to Err_{rwrel1} = 0.067668.

After multiple iterations, the final version of the optimized circuit configuration in Fig. 3(d) is obtained. The final output error value of the optimized circuit is Err_{opt} = 0.063353 and the total node count is 17. Two redundant nodes $'n9'$ and $'n10'$ are added into the configuration. Also, node $'n4'$ in Fig. 3(c) is modified such that the PI node 'F' has been pushed closer to the PO in similar lines with the first sub-case. Finally, we list some of the important differences observed in the way Boolean matching is performed by our reliabilty aware rewriting algorithm when compared to the area driven approach in [3] :

– Cuts with Primary Inputs closer to the Primary Outputs are preferred. Primary Inputs have lower error probability when compared to the internal nodes. Thus, moving a highly reliable node closer to the output can result in canceling out some of the errors.

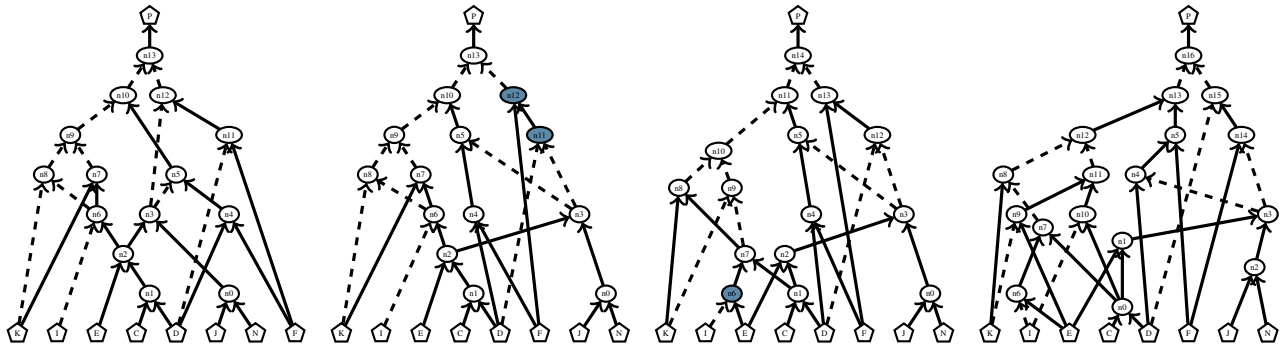
– Cones with larger circuit depth can result in higher critical path delay and are not preferable in case of delay optimisation. But, they can result in lowering the output error values and are often employed by our algorithm.

– Redundant Node insertion is commonly performed. This increases the total area but has higher masking effect and there by reducing the output node error probability.

– One other significant difference that has been observed is the matching of path lengths. Most of the time the Boolean matcher selected cuts in such a way that the resulting circuit is more balanced so that the level of both the left and right paths are similar.

B. Evaluation of Benchmark Circuits

To prove that the proposed methodology is scalable, we applied it on a set of MCNC benchmark circuits. Simulation results comparing the average circuit output errors of the original and the optimized configuration obtained from our tool are reported in Tab. I. The name of each circuit is given in the first column while column two provides the output pin count of each circuit. The following three columns list the node count of the original circuit, the optimized circuit, and the optimisation induced node count change in percentage. Columns 6 (7) provides the circuit depth of the default (optimized) circuits. Columns 8 (9) lists the average output error value for the default (optimized) circuits. Columns 10 through 12 lists the average, maximum, and



(a) Default Configuration *(b)* Config1: PI closer to PO *(c)* Config2: Redundant Node *(d)* Config3: Optimized Version

Fig. 3. MCNC Benchmark CM162A – A Case Study

the minimum relative output error improvement. Tab. II reports the area, timng and power analysis after applying RWREL on MCNC benchmark circuits. In the table, P_{org}, T_{org} and A_{org} are the power, timing and area results from the original MCNC netlist. P_{opt}, T_{opt} and A_{opt} are the power, timing and area results from the netlist obtained via RWREL optimisation.

From the tables, it is clear that significant SER reduction can be achieved very low area overhead of 6.57% that results in 13.52% higher power consumption by employing the optimisation algorithms reported in this paper. An average improvement of 14% and a peak improvement of upto 75.5% was observed. Some interesting facts from the simulation results are now discussed. $'I7'$ benchmark is a simple logic circuit consisting of 199 PI's & 67 PO's and a total node count of 903. Output pin $V266(6)$ provides the peak error improvement of 75.5%. Another important fact worth mentioning is that benchmark circuits $'I6'$, $'I7'$, $'I8'$ and $'too\_large'$ report reduction in node count. This proves that our algorithm is intelligent enough to decide when to add redundancy and when to remove unnecessary nodes that can result in better output reliability. Also benchmark circuit $'too\_large'$ reports a pretty high output error. This is because of the fact that it has 824 nodes and just 3 output pins. It implies that if large number of gates are placed within the cone to emulate a particular function, higher the probability of error on the output. Also the circuit depth of the logic resembles no direct relation to the output error as circuit depth has reduced in certain cases while in others it has been increased.

V. Conclusions and Future work

In this paper, we first described a simple gate level simulation methodology that computes circuit output error probability due to individual gate failure in complex combinational circuits. Subsequently , a reliability driven 4-cut enumeration and Boolean matching technique that improves circuit reliability has been proposed. The technique of rewriting for reliability was developed by extending an existing cut based rewriting tool to make use of local transforms targeting a reliability metric improvement instead of area. A synthesis algorithm that optimizes the circuit output nodes error probability was also presented. The application of the proposed reliability aware synthesis algorithm on various MCNC benchmark circuits with a node count from 50 to 3000 resulted in up to 75.5% output error probability reduction. On average, about 14% SER reduction was obtained at the expense of less than 4.5% area overhead. We note that the presented technique can be further improved by adopting multiple rewriting iterations, or multiple interleaved iterations of cut- and rule-based rewriting. Going forward, we intend to come up with a formal mathematical procedure that can provide a much stronger set of guidelines to the optimisation algorithm.

VI. Acknowledgement

This work was supported by the Seventh Framework Programme of the European Union, under Grant Agreement number 309129 (i-RISC project). We would also like to thank Nan Li and Dr. Elena Dubrova, KTH Sweden for helping us in resolving some of the intricate issues with the ABC tool.

References

[1] Sobeeh Almukhaizim, Yiorgos Makris, Yu-Shen Yang, and Andreas Veneris. Seamless integration of ser in rewiring-based design space exploration. In *Test Conference, 2006. ITC'06. IEEE International*, pages 1–9. IEEE, 2006.

[2] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10–16, 2005.

[3] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In *Proceedings of the 22Nd International Conference on Computer Aided Verification*, pages 24–40, 2010.

[4] Mihir R Choudhury and Kartik Mohanram. Low cost concurrent error masking using approximate logic circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(8):1163–1176, 2013.

[5] J.A. Darringer, William H. Joyner, C.Leonard Berman, and Louise Trevillyan. Logic synthesis through local transformations. *IBM Journal of Research and Development*, 25(4):272–280, 1981.

[6] Stanley L. Hurst, Jon C. Muzio, and D. Michael Miller. *Spectral Techniques in Digital Logic*. Academic Press, Inc., Orlando, FL, USA, 1985.

[7] Smita Krishnaswamy, Stephen M Plaza, Igor L Markov, and John P Hayes. Enhancing design robustness with reliability-aware resynthesis and logic simulation. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 149–154. IEEE, 2007.

[8] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *Design Automation Conference, 1997. Proceedings of the 34th*, pages 263–268, June 1997.

[9] Nan Li and E. Dubrova. Aig rewriting using 5-input cuts. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 429–430, Oct 2011.

[10] F. Mailhot and G. De Micheli. Technology mapping using boolean matching and don't care sets. In *Design Automation Conference, 1990., EDAC. Proceedings of the European*, pages 212–216, Mar 1990.

[11] A. Mishchenko and R. K. Brayton. Scalable logic synthesis using a simple circuit structure. In *Proc. IWLS*, pages 15–22, 2006.

TABLE I. RWREL PERFORMANCE EVALUATION ON DIFFERENT BENCHMARK CIRCUITS (GATE ERROR $\epsilon = 0.001$)

Benchmark	No. of Outputs	Node Count			Circuit Depth		Averaged Absolute Error		Output Error Improvement (%)		
		NC_{org}	NC_{opt}	Increase (%)	CD_{org}	CD_{opt}	Err_{org}	Err_{opt}	Avg	Max	Min
apex7	36	221	252	14.03	14	14	0.005895	0.005240	10.44	51.33	1.74
cm162a	5	36	44	22.22	9	8	0.005874	0.005128	13.88	15.66	9.74
comp	3	107	119	11.21	18	19	0.009510	0.007232	18.31	25.31	5.98
dalu	16	1371	1602	16.85	35	35	0.019144	0.016449	13.98	19.45	7.98
I6	67	692	523	-24.42	5	4	0.008381	0.007291	12.83	15.19	4.23
I7	67	903	702	-22.26	6	5	0.009235	0.008303	10.21	75.5	3.16
I8	81	3310	2187	-33.93	21	20	0.018626	0.015219	18.83	32.04	6.8
k2	45	1998	2152	7.71	23	19	0.031802	0.028530	12.94	25.65	4.52
unreg	16	112	111	-0.89	5	5	0.006506	0.005382	18.43	19.03	17.78
vda	39	924	1042	12.77	16	18	0.030577	0.027491	12.81	65.6	5.31
too_large	3	824	773	-6.19	30	27	0.053492	0.046468	13.87	17.43	11.33
8051_add	19	175	222	26.86	28	29	0.014149	0.013330	16.99	41.56	5.86
8051_mul	17	630	765	21.43	48	47	0.030946	0.029475	10.58	32.65	1.58
8051_div	17	1010	1181	16.93	198	181	0.023831	0.022888	12.31	31.75	2.18

TABLE II. AREA, DELAY AND POWER ANALYSIS – A COMPARATIVE STUDY

Benchmark	Area			Delay			Dynamic Power			Leakage Power		
	A_{org}	A_{opt}	Increase(%)	T_{org}	T_{opt}	Increase(%)	P_{org}	P_{opt}	Increase(%)	P_{org}	P_{opt}	Increase(%)
apex7	705.24	779.04	10.46	1.10	1.16	5.45	46.25	53.32	15.28	3.12	3.47	11.28
cm162a	121.68	137.88	13.31	0.69	0.56	-18.84	7.91	8.97	13.47	0.54	0.61	13.29
comp	360.36	405.72	12.59	1.36	1.38	1.47	34.22	36.11	5.54	1.61	1.79	11.46
dalu	4232.52	4876.2	15.21	3.41	3.15	-7.62	310.43	403.97	30.13	18.66	21.93	17.52
I6	1861.92	1677.6	-9.9	1.37	0.89	-35.04	133.68	137.11	2.57	8.28	7.62	-7.97
I7	2386.8	2144.16	-10.17	1.48	1.38	-6.76	174.82	175.92	0.63	10.66	9.65	-9.47
I8	8147.52	5780.16	-29.06	4.59	3.11	-32.24	720.15	522.14	-27.5	37.7	26.66	-29.29
k2	4675.32	5343.84	14.3	1.96	1.87	-4.59	119.51	177.1	48.19	23.53	26.07	10.79
unreg	367.2	372.6	1.47	0.48	0.52	8.33	29.36	32.16	9.53	1.64	1.71	4.34
vda	2158.92	2638.08	22.19	1.08	1.31	21.3	88.61	129.34	45.97	11.18	13.09	17.08
too_large	2176.2	2087.64	-4.07	2.32	2.20	-5.17	118.6	121.57	2.5	9.45	9.16	-3.08
8051_add	595.08	723.6	21.6	2.49	2.60	4.42	74.19	84.25	13.55	2.67	3.24	21.34
8051_mul	2038.32	2432.16	19.32	3.93	3.82	-2.8	304.85	363.33	19.18	9.17	10.87	18.51
8051_div	3295.8	3781.44	14.74	16.96	15.43	-9.02	555.68	612.15	10.16	14.95	17.1	14.38

[12] K. Palem and A. Lingamneni. What to do about the end of moore's law, probably! In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 924–929, 2012.

[13] R. Rithe, Jie Gu, A. Wang, S. Datla, G. Gammie, D. Buss, and A. Chandrakasan. Non-linear operating point statistical analysis for local variations in logic timing at low voltage. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 965–968, March 2010.

[14] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers. The case for lifetime reliability-aware microprocessors. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 276–287, 2004.

[15] Y. Wang, M. Enachescu, S.D. Cotofana, and L. Fang. Variation tolerant on-chip degradation sensors for dynamic reliability management systems. *Microelectronics Reliability*, 52:1787–1791, September 2012.

[16] Kai-Chiang Wu and Diana Marculescu. A low-cost, systematic methodology for soft error robustness of logic circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(2):367–379, 2013.

[17] Saeyang Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.