A Flexible Inexact TMR Technique for SRAM-based FPGAs

Shyamsundar Venkataraman, Rui Santos
Department of Electrical & Computer Engineering
National University of Singapore
Email: {shyam, elergvds}@nus.edu.sg

Akash Kumar
Center for Advancing Electronics Dresden
Technische Universität Dresden
Email: akash.kumar@tu-dresden.de

***Abstract*—Single Event Upsets (SEUs) inadvertently change the logic memory and thereby the configuration of the Field Programmable Gate Arrays (FPGAs), leading to their incorrect functioning. Traditional methods to tolerate such faults include Triple Modular Redundancy (TMR). However, such method has a high overhead in terms of power and area. Moreover, the inexact methods used in ASICs to overcome this problem are not efficient when applied in FPGAs. Therefore, this paper proposes a novel technique based on heuristic to tolerate faults in SRAM-based FPGAs by using inexact modules in conjunction with TMR, thus reducing the area and power overhead of the design. Experiments run on various MCNC benchmark circuits show the accuracy of the proposed technique. They also show that the design solutions found through this technique only differ 0.52% on average from the optimal ones and savings up to 84.4% in terms of computation time can be reached on average.**

I. Introduction

Commercial Off-The-Shelf (COTS) Static-RAM (SRAM) based Field Programmable Gate Arrays (FPGAS) have been increasingly adopted in the field of satellite and space technology since they provide rapid prototyping capabilities and quick reconfiguration ability, associated with a low cost price. However, in space the radiation from high energy particles, such as protons and neutrons, can strike the FPGA device and may upset one or more bits in the user design. These upsets, called Single Event Upsets (SEUs), inadvertently change a single or a group of bits in the FPGA, leading to an erroneous output. SEUs are traditionally masked or mitigated through a variety of techniques. One such technique is the Triple Modular Redundancy (TMR) [1] [2]. This technique triplicates the user design and the outputs of these three modules are voted upon to ensure that the errors are masked in case one of the modules fails. The primary disadvantage of TMR is the area and power overhead as it uses at least three times more area and power of the original user design, as described in Figure 1. Such high overhead might not be suitable in space environments, where both area and power are very constrained. In order to deal with these concerns, inexact computing has been considered an ideal alternative mechanism to TMR. However, the current techniques based on this mechanism are targeted to Application Specific Integrated Circuits (ASIC) technology. When they are applied to FPGAs, they may not provide the optimal solution in terms of implementation area overhead and masking factor, since the method of logic implementation differs in both. ASICs are based on logic gates and FPGA's in Look-Up-Tables (LUTs).

Key contributions: Following are the main contributions of this paper: **1)** An Inexact-TMR (ITMR) technique to protect the combinatorial circuits implemented on SRAM-FPGAs; **2)** A scalable heuristic that is not directly dependent on the size of the original circuit and gives the inexact circuit solution close to the optimal one.

Experiments ran on a various benchmarks show that the proposed heuristic only differs on average 0.14% from the Pareto front. Regarding the SER reduction, the experiments show that the design provided by the proposed heuristic only differs 0.52% on average from the optimal. Moreover, the saving of computation time can reach up 84.4% on average.

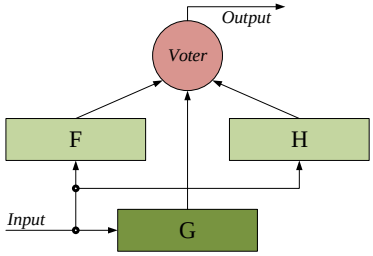

Fig. 1. TMR. Module G represents the original circuit. Modules F and H are exact replications of G.

II. Related Work

TMR [1] [2] is one of the most common fault-tolerant techniques used in COTS FPGAs. TMR triplicates the entire system and votes on the majority output. Though this technique is really good at SEU detection and fault masking, it requires a very large area and power overhead. For example, TMR requires at least thrice the area (and power) of a normal system. Various techniques have been proposed to overcome the high area and power overhead of TMR. One paper by Mohanram et al. [3] exploited the asymmetric soft error susceptibility of nodes in a logic circuit by applying two heuristics—cluster sharing reduction and dominant value reduction—to reduce the area overhead due to TMR. However, this technique modifies the synthesis of the original circuit, and still applies the conventional TMR to the clustered part of the circuit. Sierawski et al. [4] proposed a technique to mask faults in ASICs through the use of approximate modules and TMR. The approximate modules were generated by the Short-Path subsetting [5] and a good reduction of the area overhead was obtained. Similarly, Choudhury et al. [6] proposed a concurrent error detection based on approximate logic circuits to obtain fine-grained trade-offs between area overhead and the error coverage. Another paper by Sanchez et al. [7] has focused on both SEUs and Single Event Transients (SETs), the latter more prone to occur in ASICs. They have proposed the use of unate functions to approximate a given logic circuit. Though these methods are able to achieve a good fault masking and area overhead reduction, they are primarily targeted for ASICs and not FPGAs. From the discussion of the above related work, it can be concluded that TMR has inherent disadvantages when implemented in an FPGA. Moreover, the inexact computing methods used in ASICs cannot directly be applied to FPGAs since the method of logic implementation differs in both. The current inexact computing methods for ASICs try to reduce the size of the inexact circuits as much as possible, but at the same time maximizing the error output masking. In FPGAs, the circuit logic gates are implemented in LUTs. An FPGA LUT is characterized by a number of inputs and one output. It works like a block of memory that is indexed by the LUT's inputs. The output is the value stored in LUT location indexed by the inputs combination. Therefore, the number of LUTs (area

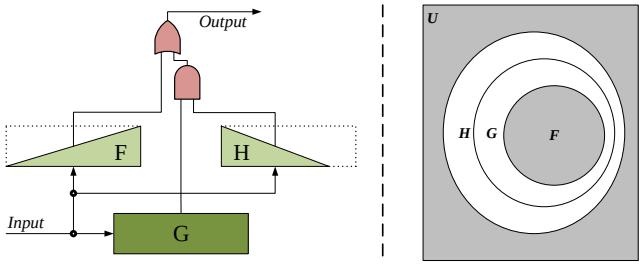

Fig. 2. **Left:** Inexact TMR. Module G represents the original circuit. Module F and H represent the approximate circuits. **Right:** Representation of the relationship among the function G, F and H. Function F is under approximated function of G. Function H is an over approximated function.

overhead) required to implement a given circuit is dependent on the number of inputs and not on the number of logic gates. In this sense, reducing the size of the circuit as much as possible in terms of logic gates, may not reduce FPGA implementation area in terms of LUTs. Bearing this in mind, a new inexact TMR (ITMR) technique is proposed in this paper. This technique based on a heuristic overcomes the area overhead of the regular TMR and it also gives better flexibility to the user to choose the inexact circuits that maximize the soft error rate reduction or according to a target (area overhead or masking factor). Moreover, it provides the inexact circuits which are closer to the optimal solution.

III. Circuit Model and Measured Parameters

As referred, in TMR (Figure 1), the modules F and H are copies of G and hence can be considered an exact copy of it. The circuit approximation technique converts circuits F and H into inexact copies of G, in such way that the outputs from the inexact modules, combined with the outputs of G would be able to effectively mask any error that occurs in F and $\overline{H}$ and some errors of G (Figure 2 – Left). With this technique, a considerable reduction in terms of implementation area and power may happen, but keeping a high level of reliability.

A. Circuit Model

Definition 1. (BOOLEAN FUNCTION) *A boolean function with n inputs can be defined as a function $f : B^n \rightarrow B$, where $B = \{0, 1\}$.*

Given the boolean function $G : B^n \rightarrow B$, the inexact functions F and H can be defined as the subset and the superset of G respectively, i.e., $F \subseteq G$ and $G \subseteq H$. This means that if an input vector $\vec{x}$ is a minterm of F, it must be a minterm of G by inclusion. Similarly, if $\vec{x}$ is a maxterm of H, it must be a maxterm of G by exclusion. This relationship is further illustrated in Figure 2 – Right. As expressions F and H tend to be closer and closer to G, the more exact the expressions become. TMR can be considered as the extreme case of this, where F and H are exactly the same as G. The voter shown in Figure 1 is replaced by the following function:

$$\hat{G} = F + (G.H). \tag{1}$$

Since the OR and AND gates are asserted by the minterms of F and maxterms of H respectively, the output $\hat{G}$ of this expression masks the faults that might occur in G.

B. Area Overhead Factor

The logic in SRAM-based FPGAs is implemented in LUTs. Thus, the area overhead can be defined as follows.

Definition 2. (AREA OVERHEAD FACTOR – A) *Area Overhead factor is a metric that defines the ratio of area used by the approximate circuits (F, H) and the original circuit G.*

Therefore, A can be represented as follows:

$$A = (\text{LUT}(F) + \text{LUT}(H))/\text{LUT}(G), \tag{2}$$

where $\text{LUT}(X)$ corresponds to the number of LUTs occupied by the circuit X after synthesis and mapping.

C. Masking Factor

When inexact circuits for F and H are used, instead of exact circuits of G, there are some input vectors in the presence of an SEU that may not produce the right value, i.e., they are not masked.

Definition 3. (MASKING FACTOR – M) *Masking Factor is a metric that defines the fraction of input vectors masked by the approximate functions F and H.*

Therefore, M can be represented as follows,

$$M = (|F| + |\overline{H}|)/2^n, \tag{3}$$

where $|F|$ expresses the number of minterms of the approximate logic function F and $|\overline{H}|$ represents the number of minterms of the disjoint function of H. Moreover, n denotes the number of inputs in the exact logic function $G : B^n \rightarrow B$.

D. Soft Error Rate (SER) Reduction

Definition 4. (SOFT ERROR RATE OF A CIRCUIT) *Soft Error Rate $SER(G)$ of a circuit G is a metric that defines the probability of G failing.*

Based on this, the SER of the $\hat{G}$ which results from the aggregation of the inexact circuits F and H to G is given by:

$$SER(\hat{G}) = (A \times SER(G)) + (1 - M) \times SER(G). \tag{4}$$

When the inexact circuits F and H are implemented, they will generate an extra implementation area A, which can also be affected by an error. Therefore, the SER for the inexact circuits can be approximated by $A \times SER(G)$. On the other hand, when the circuits F and H are implemented, the fault rate on G is reduced by the factor $(1-M)$. Moreover, the SER reduction (R) of the overall logic circuit $\hat{G}$ after the implementation of F and G, when compared to the initial SER of G is given by:

$$R = 1 - SER(\hat{G})/SER(G), \tag{5}$$

Combining Equation 4 and Equation 5, the SER reduction (R) can be further defined as:

$$R = M - A. \tag{6}$$

Please note that for the computation of R, the number of LUTs are used to measure the area overhead. This approach is one of the fairest, since the probability of the bits stored in the LUT being affected by an SEU is much smaller, comparing to the probability of the routing and control bits. For more details please refer to [8].

IV. Illustrative Example and Problem Definition

Assuming the logic function $G = \overline{x_1}.\overline{x_2} + \overline{x_1}.\overline{x_3} + \overline{x_1}.x_4$, also presented in [4], the optimal implementation of the function with 1/2-input gates in the ASIC technology requires 8 gates. From the analysis of the function in the BDD format, as well as using the mechanism that preserves the shortest paths and pruning the longest, the optimal solution for the approximation functions F and H are respectively, $F = \overline{x_1}.\overline{x_3}$ and $H = \overline{x_1}$ (Figure 3). This solution generates a masking percentage of 75%, but requires 4 logic gates to be implemented, which implies an increase on the implementation area of 50%, considering the original circuit. Regarding the FPGAs

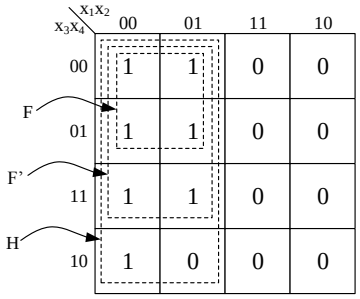

Fig. 3. Example – Karnaugh map of the logic function G.

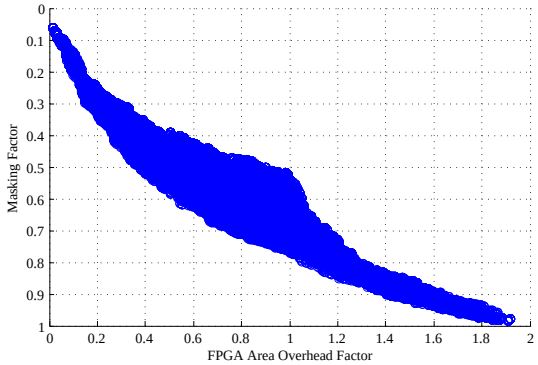

Fig. 4. Search space for the benchmark Alu4, ouput 3. It includes 78,400 different design points.

technology, we consider for this example a 3-input LUT. In this scenario, the function G requires 2 LUTs to be implemented. The inexact functions F and H can be also implemented using 2 LUTs, one for each of them. This implementation also achieves 75% of masking percentage as the previous. However, F only uses half of the space available in its corresponding LUT. Therefore, this free space can be used to increase the masking factor. For instance, $F' = \overline{x_1}.\overline{x_3} + \overline{x_1}.x_4$ (Figure 3) is the optimal solution that does not increase the area overhead in terms of LUTs, but increases the masking percentage of the logic function G. For this particular case the masking percentage reaches 87.5%.

Problem: Finding the solutions for the circuits F and H, which minimize the implementation area overhead factor and at the same time maximize the masking factor (as a result maximize the SER reduction) may be difficult and not accurate, when using the current heuristics for ASICs. Therefore, another mechanism is required for FPGAs.

V. Proposed Mechanism

Determining the inexact circuits F and H that maximize the SER reduction, through the examination of all combinations of F and H is neither practical nor scalable. For instance, Figure 4 describes the search space for the benchmark $alu4$ output 3. There are around 78,400 different inexact designs just for this particular output. Synthesizing and mapping each of these designs for FPGA require more than 3 minutes of CPU time (note this time does not include the access to the memory and I/O). In order to faster determine the optimal inexact circuits F and H, a heuristic is required. The proposed heuristic is focused on reducing the search space, in particular to restrict it to the Pareto front that identifies the optimal designs in terms of area overhead/masking factor. The proposed heuristic/algorithm receives as inputs the logic circuit G in the BDD format and the variation Δ. It provides as outputs the inexact circuits F and H that maximize the SER reduction (R). Note that the proposed heuristic can be easily reused to receive as input both a target area overhead or a masking factor, and provide as outputs the inexact design that maximizes the masking factor or minimizes the area overhead, respectively. In this proposed heuristic **four steps** can be identified and highlighted. The **first step** implements the necessary functions to obtain the main parameters that characterize the original circuit G. For that, functions provided by CUDD framework [9] are used to compute the maximum number of BDD nodes ($gNodes$), the number of inputs and the area in terms of LUTs used by the circuit G after being synthesized and mapped using the ABC Logic Synthesis Tool [10]. The **second step**, instead of synthesizing and mapping all the possible circuit combinations between F and H, synthesizes and maps independently both inexact circuits and for the sequence of nodes $[1..gNodes]$ with an interval variation given by $\lceil \Delta/100 \times gNodes \rceil$. Smaller Δs give a better range of nodes. For the obtained sequence of nodes, the circuits F and H are computed by applying the short path subsetting method to the BDD of the original circuit G. F is obtained by under approximation of G and H by over approximation. Both inexact circuits are then synthesized and mapped in order to obtain the area overhead percentage in terms of LUTs. The **third step** combines the all obtained designs for F and H in the previous section. All combinations that return unique area overhead numbers are stored. For the ones that have the same area, only the one that maximizes the number of minterms is stored. At the end of this step, the best inexact designs for the range area overhead factor $(0, 2]$ are computed. These designs are then called key designs. Finally, the **fourth step** selects among the key designs the one that maximizes the SER reduction (R). Then the corresponding F and H designs are returned. Please note that the proposed heuristic/algorithm can be easily changed to return the inexact circuit that maximizes the masking factor, according to a given target area overhead factor or, on the other hand, that minimizes the area overhead, according to a given target masking factor. For instance, this feature can be very useful if the user does not have enough space to implement the optimal solution that maximizes R. In this case, the user should choose one inexact design that maximizes the masking factor, according to the available area.

VI. Experiments and Results

This section describes the conducted experiments in order to better evaluate the proposed heuristic. They were run on Linux operating system, using a machine consisting of 8-core Intel(R) Xeon(R) processor running at a constant speed of 2.40 GHz and 8-GB of memory. The proposed heuristic was implemented in C++ and the CUDD framework was used. The logic circuits were synthesized and mapped using the ABC Logic Synthesis Tool [10] for a 6-input LUT. The feasibility of the proposed heuristic is shown based on the LGSynth93 [11] combinatorial MCNC benchmarks.

A. Heuristic Evaluation

This subsection evaluates the inexact key designs found by the proposed heuristic comparing them to the optimal ones identified by the Pareto front. Please assume the Pareto front as the optimal solution computed by the exhaustive search approach for areas overhead factor (0,2], with an interval variation equal to 0.01. Also for the same sequence of areas, the heuristic solutions are generated based on the key designs. Figure 5 presents the Pareto front (optimal points) and the points generated by the heuristic with $\Delta = 1$ and $\Delta = 10$. Note that in order to improve the readability, the Figure 5 only shows the points for the interval $A = (0.75, 1.25)$. Note also that the proposed heuristic with $\Delta = 1$ almost completely overlaps the Pareto front, highlighting its quality. For all the sequence points, the difference between the masking factor of

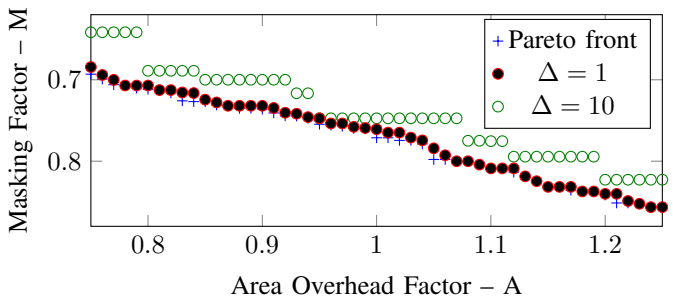

Fig. 5. Comparison between the Pareto front and the proposed heuristic for $\Delta = 1$ and $\Delta = 10$, regarding the interval $A = (0.75, 1.25)$.

TABLE I. HEURISTIC BEHAVIOUR FOR DIFFERENT ΔS (HEURISTIC APPLIED TO ALU4-OUTPUT 3).

Heuristic	*Avg. Dif.*(%)	*Max. Dif.*	*Time (sec.)*
$\Delta = 1$	0.36	0.027	0.65
$\Delta = 2$	0.84	0.039	0.37
$\Delta = 5$	2.01	0.061	0.12
$\Delta = 10$	3.39	0.083	0.10

the optimal designs and the designs given by the proposed heuristic is computed. Table I shows these detailed results for each variation (Δ = 1, 2, 5 and 10) of the proposed heuristic. Column $Avg.\ Dif.$ shows the average of the masking factor difference between the optimal designs (Pareto front) and the designs provided by the proposed heuristic, considering the sequence of areas overhead described before. Column $Max.\ Dif.$ shows the maximum measured absolute difference. Column $Time$ introduces the time required for each heuristic variation (Δ) to compute the key designs. The proposed heuristic with $\Delta = 1$, differs on average 0.36% than the optimal Pareto front. Please note that the heuristic only requires 0.65 seconds of CPU time to compute the key designs. Contrarily, the exhaustive search consumes several minutes to produce the optimal solutions. Also for the heuristic variation $\Delta = 1$, the maximum measured difference on the masking factor is 0.027. As shown in Figure 5, the proposed heuristic with $\Delta = 1$ almost completely overlaps the Pareto front. Regarding the proposed heuristic with $\Delta = 10$, it performs 3.39% worse when compared to the optimal Pareto front, but only takes 0.1 seconds to be computed. It is 6.5 times faster than the one with $\Delta = 1$. The proposed heuristic was also applied to different benchmarks for a more comprehensive assessment. Table II shows the results, for the scenario described above. Similarly to the previous table, column $Avg.\ Dif.$ shows the average of the masking factor difference between the optimal design solutions (Pareto front) and the design solutions provided by the proposed heuristic with $\Delta = 1$. Column $Max.\ Dif.$ also shows the maximum measured difference. On average the proposed heuristic with $\Delta = 1$ performs 0.14% worse than the exhaustive search, which gives the Pareto front. Note that the worst difference was measured for seq benchmark which can generate more then $1.2M$ inexact circuit combinations.

B. SER Reduction

For a set of benchmarks, Table III compares the SER reduction (R) achieved by the circuit solution provided by the heuristic ($Heur.$) and the optimal one given by the exhaustive search ($E.S.$). Column $Dif.\ R$ shows the SER reduction difference between both. The last column ($Time$) shows the computation time improvement regarding the proposed heuristic when compared to the exhaustive search. Please observe that in terms of SER reduction (R) the circuit given by the heuristic only differs 0.52% on average from the optimal one. Please also note that the proposed heuristic is on average 84.4% faster than the exhaustive approach.

TABLE II. HEURISTIC BEHAVIOUR APPLIED TO DIFFERENT MCNC BENCHMARKS ($\Delta = 1$).

Benchmark	*Avg. Dif.*(%)	*Max. Dif.*
$alu4 - o3$	0.36	0.027
$clip - o0$	0.05	0.040
$duke2 - o5$	0.00	0.001
$exp4 - o63$	0.00	0.000
$cps - o35$	0.00	0.000
$pdc - o35$	0.01	0.016
$seq - o0$	0.60	0.060
Average :	**0.14**	**0.020**

TABLE III. SER REDUCTION OF THE OPTIMAL DESIGN FOR SEVERAL MCNC BENCHMARKS.

Benchmark	*E.S. R*(%)	*Heur. R*(%)	*Dif. R*(%)	*Time*(%)
$alu4 - o3$	15.0	13.5	1.5	99.6
$clip - o0$	14.8	14.1	0.7	79.6
$duke2 - o5$	67.9	67.7	0.2	87.6
$apex4 - o63$	35.9	35.9	0.0	73.3
$cps - o35$	76.9	76.9	0.0	86.1
$pdc - o35$	73.1	72.3	0.8	65.2
$seq - o0$	76.3	75.8	0.5	99.9
Average :	-	-	**0.52**	**84.4**

VII. CONCLUSION

This paper proposes a novel technique to tolerate faults in SRAM-based FPGAs by using inexact modules in conjunction with TMR. Experiments were run using MCNC benchmarks and show that the key designs provided by the heuristic only differ 0.14% from the Pareto front. Regarding the SER reduction, the design solution provided by the heuristic only differs on average 0.52% from the optimal one. Moreover, the proposed heuristic is on average 84.4% faster than the exhaustive search mechanism.

ACKNOWLEDGMENT

This work is supported in part by the German Research Foundation (DFG) within the Cluster of Excellence Center for Advancing Electronics Dresden (cfaed).

REFERENCES

[1] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA Design Robustness with Partial TMR," in *IEEE International Reliability Physics Symposium (IRPS06)*, 2006.

[2] C. Bolchini, A. Miele, and M. D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs," in *Defect and Fault-Tolerance in VLSI Systems (DFT'07)*, 2007.

[3] K. Mohanram and N. Touba, "Partial error masking to reduce soft error failure rate in logic circuits," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03)*, 2003.

[4] B. Sierawski, B. Bhuva, and L. Massengill, "Reducing Soft Error Rate in Logic Circuits Through Approximate Logic Functions," *IEEE Transactions on Nuclear Science*, 2006.

[5] K. Ravi, K. L. McMillan, T. R. Shiple, and F. Somenzi, "Approximation and Decomposition of Binary Decision Diagrams," in *Design Automation Conference (DAC'98)*, 1998.

[6] M. R. Choudhury and K. Mohanram, "Approximate Logic Circuits for Low Overhead, Non-Intrusive Concurrent Error Detection," in *Design, Automation and Test in Europe Conference (DATE'08)*, 2008.

[7] A. Sanchez-Clemente, L. Entrena, M. Garcia-Valderas, and C. Lopez-Ongil, "Logic masking for SET Mitigation Using Approximate Logic Circuits," in *IEEE Int. Conf. On-Line Testing Symp. (IOLTS'12)*, 2012.

[8] P. Graham, M. Caffrey, J. Zimmerman, and D. Eric Johnson, "Consequences and Categories of SRAM FPGA Configuration SEUs," in *International Conference on Military and Aerospace Programmable Logic Devices (MAPLD'03)*, 2003.

[9] "CU Decision Diagram Package ," http://http://vlsi.colorado.edu/ fabio/CUDD/.

[10] "ABC Logic Synthesis Tool," http://www.eecs.berkeley.edu/ alanmi/abc/.

[11] "MCNC Benchmarks," http://http://ddd.fit.cvut.cz/prj/Benchmarks/.