IPSJ Transactions on System LSI Design Methodology Vol.13 35–38 (Feb. 2020)

[DOI: 10.2197/ipsjtsldm.13.35]

Short Paper

A Logic Optimization Method by Eliminating Redundant Multiple Faults from Higher to Lower Cardinality

Peikun Wang[1,a)] Amir Masaud Gharehbaghi[2,b)] Masahiro Fujita[2,c)]

Received: June 4, 2019, Revised: August 26, 2019,
Accepted: October 27, 2019

Abstract: In this paper, we propose a logic optimization method to remove the redundancy in the circuit. The incremental Automatic Test Pattern Generation method is used to find the redundant multiple faults. In order to remove as many redundancies as possible, instead of removing the redundant single faults first, we clear up the redundant faults from higher cardinality to lower cardinality. The experiments prove that the proposed method can successfully eliminate more redundancies comparing to the redundancy removal command in the synthesis tool SIS.

Keywords: logic optimization, redundant multiple faults, incremental fault selection method

1. Introduction

Logic optimization methods have been studied since 1980's which resulted in SIS [1] logic synthesis tool and later in ABC [2] logic synthesis and verification tool. Redundancy removal command in logic synthesis tool SIS [1] can optimize the circuit by eliminating the single redundant faults. Authors in Ref. [3] try to remove the single redundancies in the circuit by repeatedly adding redundancy and then removing other redundancies. Authors in Ref. [4] utilize multiple faults to simplify circuit logics by generating a new circuit with the function approximate to the original circuit. In other words, the circuit function may be changed. However, none of these methods consider the optimization by removing the redundant multiple faults in the circuit without the function change, which can potentially eliminate more redundancies and make circuit more compact.

In this paper, we propose a new logic optimization method by identifying the redundant multiple stuck-at faults (MSAF) and removing their related gates. The fault selection method introduced in Refs. [5], [6], [7] is applied to find the redundant multiple faults. According to the experimental results, by eliminating the MSAF from higher cardinality to lower cardinality, the proposed method can remove more redundant logic comparing to the methods that only eliminate single stuck-at faults (SSAF).

The rest of the paper is organized as follows. Section 2 presents the fault selection method to find multiple faults. Section 3 explains the proposed fault removal method. Section 4 illustrates the experimental results. Section 5 concludes the paper and discusses the future work.

[1] Electrical Engineering and Information Systems, The University of Tokyo, Bunkyo, Tokyo 113–0032, Japan
[2] VLSI Design and Education Center, The University of Tokyo, Bunkyo, Tokyo 113–0032, Japan
a) paykoon@cad.t.u-tokyo.ac.jp
b) amir@cad.t.u-tokyo.ac.jp
c) fujita@ee.t.u-tokyo.ac.jp

2. Finding the Redundant Multiple Faults

The naive way to find redundant faults is traversing the entire fault list and checking one by one. However, it cannot handle multiple faults due to the large number of fault combinations. Inspired by the research [5], [6], [7], instead of checking all multiple faults, we want to check the fault propagation path to find the redundant multiple fault, which can greatly reduce the time, as shown in the experimental results of Refs. [5], [6], [7]. Here we take **Fig. 1** as an example. We assume that there are two SSAF f_1 and f_2. f_1 cannot be propagated to the output since f_2 blocks its propagation path. If f_2 is a redundant fault or f_1 blocks the path of f_2 as well, Double Stuck-at Faults (DSAF) $\{f_1, f_2\}$ is selected as a potential redundant fault. In addition, the DSAF consists of two redundant single faults is also classified as the potential redundant fault. Then, we can pick up the redundant DSAF by performing fault simulation and test generation. This process is very fast since the size of the potential redundant fault list is drastically smaller than all the faults. Therefore, starting from a compact test set for the single fault, we can incrementally find the redundant multiple faults in an acceptable running time for fairly large circuits.

3. Removal of the Multiple Faults

The composition of the redundant MSAF is shown in **Table 1**,

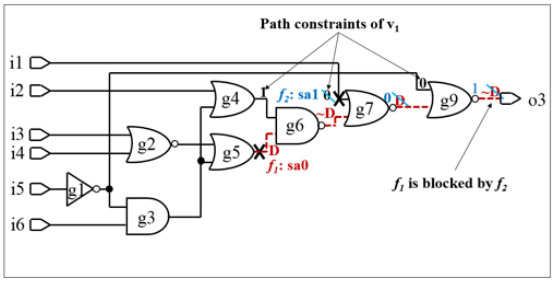

Fig. 1 Two faults mutually block each other [5].

whose second, third, and forth columns are the number of the redundant DSAFs including two non-redundant SSAFs, one non-redundant and one redundant SSAFs, and two redundant SSAFs, respectively. The redundant DSAF consisting of two redundant SSAFs is equal to two SSAFs as redundant faults, while the DSAF include at least one non-redundant SSAF is different from two redundant SSAFs. Obviously, most of the redundant DSAFs include at least one redundant SSAF. There are two ways to remove the redundant multiple faults. The first way is to remove the faults from the lower cardinality to higher cardinality. However, according to the experimental results shown in Table 1, few redundant multiple faults such as double faults can be found if we remove all the single faults at first, because most of the redundant multiple faults include at least one redundant single fault. In other words, if we clear all redundant single faults, most of the original redundant multiple faults become non-redundant; hence, we cannot remove them to optimize the circuit structure. In contrast, if we remove the redundant faults from higher cardinality to lower cardinality, we can remove the redundant single fault as well as the non-redundant single fault that are included in the redundant multiple faults, which means that we can make the circuit smaller in size and more compact.

Similarly, in order to optimize more redundancies, in the logic optimization process of the MSAFs with a same cardinality, the MSAF with more non-redundant SSAFs is removed first. Since many MSAFs include same redundant SSAFs, if we remove the MSAF with more redundant SSAFs first, many of the redundant MSAFs with non-redundant SSAFs is not redundant any more, which decreases the number of redundancies to be removed.

Table 1 Composition of the redundant DSAF.

Circuit	Two NoRe-SSAF	NoRe-SSAF+Re-SSAF	Two Re-SSAF	Total Re-DSAF
S444	0	21	30	51
S832	0	11	35	46
S1238	0	103	1,583	1,686
S1423	0	36	36	72
S1494	0	19	17	36
S5378	0	80	227	307
S9234	0	1,579	16,994	18,577
S35932	0	4,128	437,248	441,376
S38417	0	20,507	95,983	116,490

4. Experimental Results

We use the Glucose 4.1 [9] as the SAT solver to generate the test patterns and find redundant faults. We perform the experiment with ISCAS 89 [10] and IWLS 2005 [11] benchmark circuits. The fault selection of the ISCAS 89 circuits starts from the compact test set for the SSAF [12]. The single test patterns of the IWLS 2005 circuits are generated by a commercial tool.

In **Table 2**, the experiment results illustrated from the second column to the sixth column are the number of the gates and the redundant SSAF and DSAF. The second column is the number of gates in the circuit. The third column is the number of all redundant SSAFs. The fourth column is the number of the selected redundant DSAF if we do not remove the redundant single fault in the circuit. The fifth column is the number of selected redundant double faults after we remove all redundant single fault in the circuit. Obviously, no redundant double faults is found if we clear all single redundancy, because most of the redundant double faults include at least one redundant single fault in these circuits. The sixth column is the number of redundant DSAF that actually used to optimize the circuit. Many redundant DSAFs may include the same redundant or non-redundant SSAF, which means that only a portion of DSAFs can already cover all the gates we can optimize. Consequently, the actual number of DSAF used to eliminate the circuit redundancy is much smaller than the total number of DSAF.

The number of removed gates by deleting the faults is shown in seventh and eighth columns. The seventh column is the number of gates that can be removed by optimizing the redundant single fault using the redundancy removal command in the logic synthesis tool SIS [1]. The eighth column illustrates the number of gates

Table 2 Number of redundant faults and gates removed by eliminating redundancy.

Circuit	Gate	Re-SSAF	Re-DSAF (Has Re-SSAF)	Re-DSAF (No SSAF)	Used Re-DSAF	Re-SSAF Removed Gates	Re-DSAF Removed Gates	Total Runtime (minute)
S444	212	18	51	0	9	12	13	0.003
S832	404	12	46	0	6	9	9	0.008
S1238	597	84	1,686	0	44	60	60	0.45
S1423	635	28	72	0	8	16	16	0.03
S1494	712	15	36	0	6	11	11	0.8
S5378	1,912	56	307	0	15	23	23	0.16
S9234	2,534	332	18,577	0	130	185	201	4.5
S35932	16,047	10,688	441,376	0	1,920	2,304	2,560	35
S38417	16,047	154	441	0	87	134	139	3.1
S38584	16,009	2,037	116,490	0	651	841	865	13
des_area	5,545	16	44	0	6	12	13	14.9
systemcaes	14,831	282	3,804	0	142	177	204	200
usb_funct	19,752	633	6,747	0	218	336	344	5.5
wb_conmax	52,658	8,918	221,264	0	3,127	4,786	5,100	22.3

removed by eliminating the selected DSAF first and then remove the SSAF. Notice that the number of redundant SSAF and DSAF in the third and fourth columns are larger than the removed gates in seventh and eighth columns, respectively, because some of the redundant SSAFs locate in the same gate. In addition, although the number of the redundant DSAF is one or two order of magnitudes larger than the SSAF, the improvement ratio is not that much, because most of the DSAFs consist of two redundant single faults, and only a few of the DSAFs include a non-redundant SSAF. Therefore, in most of the cases, the removal of the redundant DSAF is to remove the redundant SSAF. In the smaller size circuits, such as S444, S832 and S1238, the number of the removed gates by optimizing the single and double redundancy shown in seventh and eighth columns are almost the same, since most of the DSAFs in those circuits consist of only the redundant SSAF. Therefore, if we eliminate all the redundant DSAFs, the redundant SSAFs are eliminated as well. In contrast, in larger size circuits such as systemcase, 15% more gates can be removed if we eliminate the DSAF first, since many of the redundant DSAFs in those circuits include the non-redundant SSAF. In addition, we have performed the experiment by removing the double faults in every possible combinations. However, the number of eliminated gates are smaller than the results in Table 2. It is because that the proposed method removes the redundant DSAF including a non-redundant SSAF first, which guarantees that more redundancies can be eliminated. The experimental results illustrate that the proposed method is capable of optimizing the circuit and removing more redundancies following the proposed heuristic of the fault removal.

The total running time is shown in the ninth column. Actually, fault selection process of the redundant MSAFs takes most of the running time. Once we obtain the redundant fault list, the optimization process can be completed very fast. Most of the circuits can finish the entire optimization process in an acceptable time. Notice that, the circuit such as systemcase takes more than 1 hour to finish the redundant fault selection process, since our method are based on SAT-solver, which is not good at handling the circuit with a large number of XOR gates [5], [6], [7].

5. Conclusion

In this paper, we have proposed a logic optimization method by first identifying and then removing the redundant multiple faults. The incremental ATPG method is used to pick up the redundant multiple faults. By eliminating the redundant faults from higher cardinality to lower cardinality, more redundant logics can be optimized. The experimental results show that the proposed method can remove more redundancies comparing to the redundancy removal command of SIS, which proves the feasibility of the proposed method. Moreover, the experimental results prove that the proposed method can finish the process of the redundant fault selection and removal within an acceptable time. Our future work is to make our implementation more efficient and utilize higher cardinality of faults, such as redundant triple faults, in the optimization, and improve the processing speed of the fault selection process.

References

[1] Sentovich, E.M. et al.: SIS: A system for sequential circuit synthesis, *Memorandum No.UCB/ERL M92/41* (1992).

[2] Brayton, R. and Mishchenko, A.: ABC: An academic industrial-strength verification tool, *International Conference on Computer Aided Verification*, Springer, Berlin, Heidelberg (2010).

[3] Entrena, L.A. and Cheng, K.-T.: Combinational and sequential logic optimization by redundancy addition and removal, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.14, No.7, pp.909–916 (1995).

[4] Shin, D. and Gupta, S.K.: A new circuit simplification method for error tolerant applications, *2011 Design, Automation & Test in Europe*, IEEE (2011).

[5] Wang, P. et al.: An ATPG method for double Stuck-At faults by analyzing propagation paths of single faults, *IEEE Trans. Circuits and Systems I: Regular Papers*, Vol.65, No.3, pp.1063–1074 (2017).

[6] Wang, P., Gharehbaghi, A.M. and Fujita, M.: Automatic Test Pattern Generation for Double Stuck-at Faults Based on Test Patterns of Single Faults, *20th International Symposium on Quality Electronic Design* (*ISQED*), IEEE (2019).

[7] Wang, P., Gharehbaghi, A.M. and Fujita, M.: An Incremental Automatic Test Pattern Generation Method for Multiple Stuck-at Faults, *The 37th International VLSI Test Symposium* (*VTS*), IEEE (2019).

[8] Kim, Y.C., Saluja, K.K. and Agrawal, V.D.: Multiple faults: Modeling, simulation and test, *Proc. 2002 Asia and South Pacific Design Automation Conference*, IEEE Computer Society (2002).

[9] Audemard, G. and Simon, L.: GLUCOSE: A solver that predicts learnt clauses quality, *SAT Competition*, pp.7–8 (2009).

[10] Brglez, F., Bryan, D. and Kozminski, K.: Combinational profiles of sequential benchmark circuits, *IEEE International Symposium on Circuits and Systems*, Vol.3 (1989).

[11] Albrecht, C.: IWLS 2005 benchmarks, *International Workshop for Logic Synthesis* (*IWLS*) (2005), available from ⟨http://www.iwls.org⟩.

[12] Eggersglüb, S. et al.: Optimization-based multiple target test generation for highly compacted test sets, *2014 19th IEEE European Test Symposium* (*ETS*), IEEE (2014).

Peikun Wang received his B.S. degree in Electronic Engineering, and his M.S. degree in Communication and Information Systems from South China University of Technology, Guangzhou, China, in 2013 and 2016, respectively. He is currently pursuing his Ph.D. degree in Electrical Engineering and Information Systems at The University of Tokyo, Tokyo, Japan. His research interests include automatic test pattern generation technology in VLSI design, and the digital hardware design based on FPGA.

Amir Masoud Gharehbaghi received his Ph.D. in computer engineering from Sharif University of Technology in 2007. He is currently a researcher in the VLSI Design and Education Center (VDEC), The University of Tokyo, Japan. Previously, he was project assistant professor in the Department of Electrical Engineering and Information Systems, The University of Tokyo, Japan, and postdoctoral researcher in the VLSI Design and Education Center (VDEC), The University of Tokyo, Japan. His research interests include design and verification techniques for embedded system, IoT, and AI accelerators.

Masahiro Fujita received his Ph.D. in Information Engineering from the University of Tokyo in 1985 on his work on model checking of hardware designs by using logic programming languages. In 1985, he joined Fujitsu as a researcher and started to work on hardware automatic synthesis as well as formal verification methods and tools, including enhancements of BDD/SAT-based techniques. From 1993 to 2000, he was director at Fujitsu Laboratories of America and headed a hardware formal verification group developing a formal verifier for real-life designs having more than several million gates. The developed tool has been used in production internally at Fujitsu and externally as well. Since March 2000, he has been a professor at VLSI Design and Education Center of The University of Tokyo. He is currently the director of the VLSI Design and Education Center of The University of Tokyo. He has done innovative work in the areas of hardware verification, synthesis, testing, and software verification-mostly targeting embedded software and web-based programs. He has been involved in a Japanese governmental research project for dependable system designs and has developed a formal verifier for C programs that could be used for both hardware and embedded software designs. The tool is now under evaluation jointly with industry under governmental support. He has authored and co-authored 10 books, and has more than 200 publications. He has been involved as program and steering committee member in many prestigious conferences on CAD, VLSI designs, software engineering, and more. His current research interests include synthesis and verification in SoC (System on Chip), hardware/software co-designs targeting embedded systems, digital/analog co-designs, and formal analysis, verification, and synthesis of web-based programs and embedded programs.

(Recommended by Associate Editor: *Satoshi Ohtake*)