Improving Redundancy Addition and Removal Using Unreachable States for Sequential Circuits

Xiaoqing Yang, Zigang Xiao, Y. L. Wu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Email: {xqyang, zgxiao, ylw}@cse.cuhk.edu.hk

***Abstract*— Redundancy Addition and Removal (RAR), one of the major combinational logic perturbation techniques, has been shown to be very useful for many EDA optimization tasks. However, all the currently known RAR techniques did not analyze and make use of unreachable states, which are abundant in sequential circuits. These unreachable states can be considered as input don't cares and can add an extra flexibility in locating alternative wires. In this paper, we study the fundamental theory and propose a reasoning scheme for locating alternative wires without performing wasteful redundancy tests. To explore the deeper effect of unreachable states, the concept is extended to illegal assignments and the fault independent redundancy identification is applied on illegal assignments to find flexibilities introduced by unreachable states. On the experiments carried for both MCNC and industry benchmarks, it is shown that using such an idea, a remarkable increase of more than 100% (averagely) in the number of alternative wires can be found, which should be quite useful as most of today's practical circuits are sequential.**

I. Introduction

Rewiring is a technique to reconnect logic wires in a Boolean network without changing the functionality of the circuit. This technique has been widely applied for many EDA optimization problems, such as circuit optimization [1]–[6], partitioning [7] and FPGA synthesis [8] [9], [10].

Redundancy addition and removal [1]–[6], [11], [12] is an ATPG-based rewiring technique, which adds redundant wires or gates to a circuit so that other wires or gates may become redundant and removable. RAMFIRE [11] proposed a single-pass rewiring approach that utilized a scheme FIRE [13] to directly identify alternative wires for target wire without trial-and-error redundancy tests. In a recently proposed SAT-controlled RAR scheme [14] using Boolean constraint propagation, a conflict-driven learning and flexible decision procedure can identify 36% more of alternative wires than that produced by the previous 2-way RAR without recursive learning.

Most of the known rewiring techniques explore its capacity under a pure context of combinational Boolean networks only. However, though rewiring is an effective logic perturbation technique for combinational logics, a combinational logic is rarely housed inside a chip without sequential devices.

Entrena *et al.* provided redundancy addition and removal for sequential logic [2]. Implication of mandatory assignments may occur at different time frames from the target wire. It is possible to add alternative wires across time frames by inserting flip-flops. [15] proposed a transformation scheme by adding redundancies to block initialization of other wires. In this approach, it is necessary to apply a costly multi-fault model on sequential redundancy identifications and the number of necessary time frames might be quite large.

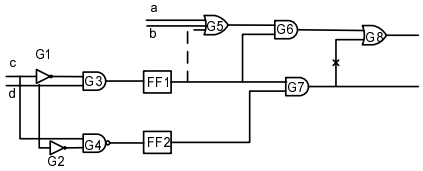


Fig. 1. Example for Rewiring in Sequential logic

In [16], Yuan and Xu developed an efficient algorithm to identify illegal states for pseudo-functional testing in gate level circuits. They showed that the main structural source causing illegal states is the multi-fanout topology of the circuit. Based on the illegal states in pseudo-functional testing found by [16], we will explore further on the theories and flexibility of rewiring under the context of sequential circuits. We will give a new technique of redundancy identification by utilizing the information of unreachable states in a sequential logic. All alternative wires found by this paper are in the same time frame with the corresponding target wire. This technique does not require insertions of any flip-flop for alternative wires.

Our experimental results for a set of MCNC and industry benchmarks showed that, with this extra flexibility considered for sequential machines, the number of alternative wires found is quite significantly increased for over 100% on average.

The remainder of this paper is organized as follows. Section II briefly motivates this paper. Section III gives the idea of extending concept of unreachable states. Section IV explains how unreachable states can be used to enhance rewiring scheme. The experimental results are shown in Section V. Section VI summarizes the contribution of our work.

II. Motivation

Some states in a sequential circuit cannot be reached by any input sequence. A state is said to be unreachable if it has no transition from any other state. If we use the information of unreachable states in sequential circuits, more alternative wires can be found because some test patterns originated from unreachable states will not happen.

Fig. 1 illustrates an example of rewiring in a sequential circuit and Fig. 2 shows a part of the combinational logic as

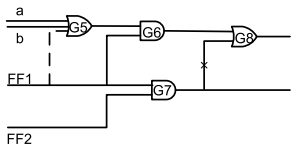
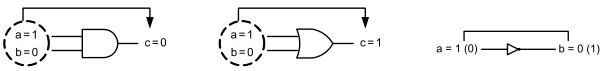


Fig. 2. Combinational Logic Part in Fig. 1



(a) rule for AND (b) rule for OR (c) rule for NOT

Fig. 3. Forward Implication Rule

in Fig. 1. Although adding wire ($FF_1 \rightarrow G_5$) can make target wire $w_t(G_7 \rightarrow G_8)$ redundant, wire ($FF_1 \rightarrow G_5$) itself is not redundant since the test vector (0,0,1,0) can test the stuck-at-0 fault on it. Suppose we know that ($FF_1 = 1, FF_2 = 0$) is an unreachable state in Fig. 1, the wire ($FF_1 \rightarrow G_5$) st-0 fault will be untestable and it becomes an alternative wire for target wire $w_t(G_7 \rightarrow G_8)$ in the sequential logic.

Definition 1: w_a is *sequential redundant* if w_a is redundant in the sequential circuit no matter whether it is redundant in the combinational logic part.

III. Redundancy Identification using Unreachable States

An unreachable state can be treated as a conflict combination in the theory of FIRE [13]. An unreachable state $\{FF_1 = v_1, FF_2 = v_2, \ldots FF_n = v_n\}$ means that this assignment is illegal in the circuit. This combination can be treated as a conflict assignment. By propagating $\{FF_1 = \overline{v_1}, FF_2 = \overline{v_2}, \ldots FF_n = \overline{v_n}\}$, the stuck-at faults which are blocked by this unreachable state can be found.

Reference [17] provides a backward implication technique for the illegal combination before propagating unobservability and uncontrollability. The technique helps to find more stuck-at faults. But backward implication for unreachable state is useless for rewiring because unreachable state is generated by confliction in previous time frame.

However, unreachable states and all illegal assignments can be processed by forward implication. This forward implication also helps to find more stuck-at faults.

Fig. 3 illustrates the forward implication rules. In Fig. 3(a), if $\{a = 1, b = 0\}$ is an unreachable state or illegal combination, $\{a = 1, c = 0\}$ is also an illegal combination (unreachable combination). According to Theorem 1, $\{a = 1, b = 0\}$ can be replaced by $\{a = 1, c = 0\}$. In Fig. 3(b), if $\{a = 1, b = 0\}$ is an unreachable state or illegal combination, $\{a = 1, b = 0\}$ can be replaced by $\{b = 0, c = 1\}$.

Theorem 1: Given a set of illegal assignments or unreachable states *V*, the set of illegal assignments *V'* propagated by using the rules illustrated in Fig. 3 can always cover those found by *V*.

Proof: For the forward implication rule on a AND gate illustrated in Fig. 3(a),

$$c = \overline{0} \rightarrow b = \overline{0}$$
$$\Rightarrow S(b = \overline{0}) \subseteq S(c = \overline{0})$$
$$\Rightarrow \left\{S(b = \overline{0}) \cap S(a = \overline{1})\right\} \subseteq \left\{S(c = \overline{0}) \cap S(a = \overline{1})\right\}$$

So, the illegal assignment {a=1, b=0} can be replaced by {a=1, c=0} to search more alternative wires. Due to page limitation, proofs for Fig. 3(b) and Fig. 3(c) are omitted.

Consider the example shown in Fig 1, suppose we want to find the redundant faults by using an unreachable state $\{FF_1 = 1, FF_2 = 0\}$. $S(FF_1 = \overline{1})$ and $S(FF_2 = \overline{0})$, the set of assignments generated from unreachable state, should be computed. We assign $FF_1 = \overline{1}$ and process the implications according to the rules introduced in [11] [13]. Since FF_1 cannot be 1, G_6 and G_7 cannot be 1 and are assigned by $\overline{1}$, similarly we have $\overline{1}$ on G_8. Any stuck-at fault on $FF_2 \rightarrow G_7$ needs FF_1 to be 1, therefore $FF_1 = \overline{1}$ implies FF_2 unobservable. G_5 is unobservable because any stuck-at fault on $G_5 \rightarrow G_6$ needs FF_1 to be 1. Using the backward implication rule of unobservability, we know that a, b and $FF_1 \rightarrow G_5$ are unobservable. Hence we have $S(FF_1 = \overline{1}) = \{FF_1 = \overline{1}, FF_2 = *, G_7 = \overline{1}, G_8 = \overline{1}, G_6 = \overline{1}, G_5 = *, a = *, b = *\}$. For $FF_2 = \overline{0}$, no implication can be made. So, we get $S(FF_2 = \overline{0}) = \{FF_2 = \overline{0}\}$. The set of stuck-at faults that require FF_1 to be 1 to be detected is $F(FF_1 = \overline{1})$ = $\{FF_1 \rightarrow G_5$ st-0, $FF_2 \rightarrow G_7$ st-1, $a \rightarrow G_5$ st-0, $b \rightarrow G_5$ st-0, $G_5 \rightarrow G_6$ st-1, $G_6 \rightarrow G_8$ st-0, $G_7 \rightarrow G_8$ st-0$\}$. The set of stuck-at faults that need FF_2 to be 0 is $F(FF_2 = \overline{0})$ = $\{FF_2 \rightarrow G_7$ st-1$\}$. Obviously, $F(FF_1 = \overline{1}) \cap F(FF_2 = \overline{0})$ = $\{FF_2 \rightarrow G_7$ st-1$\}$.

If the unreachable state $\{FF_1 = 1, FF_2 = 0\}$ is propagated by the forward implication rule in Fig. 3, the new inconsistent assignment is $\{FF_1 = 1, G_7 = 0\}$. We assign $G_7 = \overline{0}$ and process implications to get $S(G_7 = \overline{0})$. Because G_7 cannot be 0, either input of G_7 cannot be 0. So, both FF_1 and FF_2 are assigned by $\overline{0}$. G_8 cannot be 0 due to its fanin G_7. Given that any stuck-at fault on $G_6 \rightarrow G_8$ needs G_7 to be 0, $G_7 = \overline{0}$ implies G_6 unobservable. Using the backward implication rule for unobservability, we know that all of G_5, a and b are unobservable. Hence, we have $S(G_7 = \overline{0}) = \{FF_1 = \overline{0}, FF_2 = \overline{0}, G_7 = \overline{0}, G_8 = \overline{0}, G_6 = *, G_5 = *, a = *, b = *, (FF_1 \rightarrow G_5) = *\}$. The set of stuck-at faults that needs G_7 to be assigned as 0 is $F(G_7 = \overline{0})$ = $\{a \rightarrow G_5$ st-0, $b \rightarrow G_5$ st-0, $FF_1 \rightarrow G_5$ st-0, $FF_1 \rightarrow G_7$ st-1, $FF_2 \rightarrow G_7$ st-1, $G_5 \rightarrow G_6$ st-1, $FF_1 \rightarrow G_6$ st-1, $G_6 \rightarrow G_8$ st-0$\}$. So, $F(FF_1 = \overline{1}) \cap F(G_7 = \overline{0})$ = $\{FF_1 \rightarrow G_5$ st-0, $FF_2 \rightarrow G_7$ st-1, $a \rightarrow G_5$ st-0, $b \rightarrow G_5$ st-0, $G_5 \rightarrow G_6$ st-1, $G_6 \rightarrow G_8$ st-0$\}$. More redundant faults can be found by utilizing the forward implication rule described by Fig. 3.

IV. Alternative Wire Identification

This section describes the process of finding alternative wires by utilizing the information of unreachable states. The process includes three main steps:

- Use forward implication rule to extend unreachable states. Launch FIRE to get the sets of value assignments resulting from the extended unreachable assignments.
- For a given target wire w_t, identify candidate alternative wires which, when added into the circuit, will make w_t redundant.

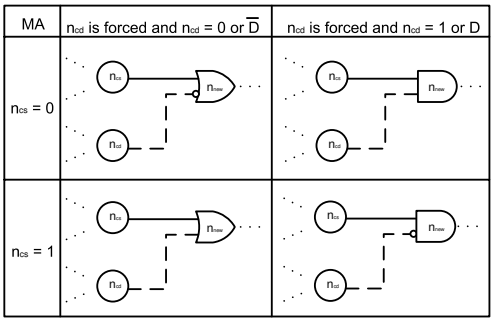


Fig. 4. Rule for Adding New Gate

- Verify whether each candidate alternative wire is a real alternative wire.

For a given target wire w_t with source gate n_s and sink gate n_d, different stuck-at fault tests will be launched on w_t according to the type of n_d. If n_d is an AND gate or a NAND gate, we will perform st-1 test on w_t. Instead, if n_d is an OR gate or a NOR gate, we will perform st-0 test on w_t. After performing the stuck-at fault test, a set of mandatory assignments (MA) [12], which contains some forced MAs (FMA) [12], would be found. Based on the MAs and Forced MAs, we can find candidate alternative wires which when added, can make w_t redundant. Gates with MA can be candidate wire's source gate, denoted by n_{cs}. Gates in FMA can be the candidate wire's sink, denoted by n_{cd}.

According to [12] and [17], we can get a rule for adding a new gate n_{new} with the input n_{cd} and n_{cs} to make the target wire redundant. The rule is shown in Fig. 4. [17] shows the relationship between redundancy and potential redundancy. We expand the method to the sequential redundancy field.

Given a sequential circuit C, a candidate wire w_c whose source gate is n_{cs} and sink gate is n_{cd}, a set of implication result S generated by FIRE on unreachable assignments, we have the following definitions:

Definition 2: The source gate n_{cs} is *sequential potentially redundant* if n_{cs} has a MA value of 0(1) when doing the stuck-at fault test on w_t and n_{cs} gets a value of $\overline{0}(\overline{1})$ in S.

Definition 3: The sink gate n_{cd} is *sequential potentially redundant* if either of the following conditions is satisfied:

- n_{cd} gets a value * in S;
- n_{cd} has a MA value of 0(1) when doing the stuck-at fault test on w_t and n_{cd} gets a value $\overline{0}(\overline{1})$ in S.

Definition 4: Wire w_c is *sequential potentially redundant* in S if either its source or its sink is sequential potentially redundant in S.

Fig. 5 outlines how to find alternative wire(s) in sequential circuits by utilizing the unreachable states. In our algorithm, we first expand unreachable states to unreachable assignment sets to find more stuck-at faults. Each assignment in the unreachable assignment sets would be propagated under the rule of uncontrollability and unobservability [13]. For a given target wire, stuck-at fault test on this wire would be performed to construct candidate alternative wires. All these candidate alternative wires, when added into the circuit, would make target wire redundant. At last, for each candidate alternative wire, we will check whether it is sequential redundant if it is added into the circuit. All candidate alternative wires which satisfy this condition would be alternative wire of the given target wire.

Algorithm 1 FindRelativeAW(Circuit C, Unreachable States U, target wire w_t)

```
1: expand unreachable states U to unreachable assignment sets A
2: for each unreachable assignment A_i in A do
3:    for each assignment x_j=v_j in A_i do
4:       find S_i(x_j = v_j' )
5:    end for
6: end for
7: perform stuck at fault test on w_t in C
8: construct candidate alternative wires W according to Figure 4
9: AW = ∅
10: for each candidate alternative wire w_c in W do
11:    if w_c is relative redundant in any S_i then
12:       AW = AW ∪ {w_c}
13:    end if
14: end for
15: return AW
```

Fig. 5. Outline of Sequential Alternative Wire Identification

V. Experimental Results

We implemented our tool in C++ language. Experiments are carried on a Intel Pentium IV 2.80 GHz 1.0GB RAM machine.

In the first experiment, to evaluate the effectiveness of our solution, we perform experiments on MCNC benchmarks, comparing against REWIRE [12]. In the experiment, we found that about half of the benchmarks which have unreachable states found by [16] had no improvements on the number of alternative wires by our method. However, as shown by Table I, to the other circuits which have gains with our algorithm, more than 100% alternative wires can be found, when compared to [12]. These alternative wires are verified by *sec* command in ABC [18]. The first column gives names of circuit. The number of alternative wires found by [12] and CPU times are shown in the second and the third columns. The fourth and the fifth columns show the number of unreachable cubes found by [16] and the corresponding CPU time. The following two columns give the number of alternative wires found by our method and the corresponding CPU time. The last two columns compare the number of alternative wires and CPU time of the methods respectively. On average, our method obtains about 150% (249.4%-100%) improvements in the number of alternative wires found with about 900%(1057.78%-100%) increase on CPU time. Actually, with s420 and s5378 excluded, the CPU time increase will be decreased to no more than 300% for the other benchmarks.

In the second experiment, we perform experiments on industry benchmark circuits. Although about half of the benchmarks possessing unreachable states have no improvements on the number of alternative wires, our algorithm can still find more than 100% of alternative wires on the remaining circuits, when compared to [17]. Table II shows that there is a 131% (231.39%-100%) improvement in the number of alternative wires found if the unreachable states are considered. The penalty on CPU time is about 60%. Some benchmarks can achieve surprising results due to their having more applicable unreachable states, which are highly circuits dependent.

The above experiments demonstrate that our method can lead to significant improvements in terms of the number of

TABLE I

COMPARISON ON MCNC BENCHMARKS

Circuit	REWIRE [12]		unreachable states [16]		Our method		Ratio on	Ratio on
	alt wires	CPU time (sec)	unreachable cube	CPU time (sec)	alt wires	CPU time (sec)	alt wires(%)	CPU time (%)
b02	189	0.01	4	0.01	193	0.02	102.12	200
b03	476	0.2	114	0.08	480	1.01	100.84	505
b04	1530	2.33	1	0.24	1536	16.66	100.39	715.02
b05	10073	8.18	170	0.52	10118	57.89	100.45	707.7
b09	1133	0.3	1	0.04	1145	1.06	101.06	353.33
b11	3458	7.22	3	0.17	3777	19.07	109.22	264.13
s208	408	0.17	16	0.04	1645	1.22	403.19	717.65
s386	6439	1.33	7	0.05	6440	7.13	100.02	536.09
s420	1235	1.25	48	0.11	10021	39.34	811.42	3147.2
s5378	11246	11.22	4321	17.71	45391	887.06	403.62	7906.06
alu2	3518	5.78	4829	38.03	8285	9.74	235.5	168.51
alu4	2804	3.78	15018	247.99	8623	7.68	307.52	203.17
apex3	6085	152.34	37095	397.60	39347	277.41	646.62	182.1
apex5	2656	14.58	1739	34.96	2683	19.06	101.02	130.73
apex6	1482	1.5	670	14.22	1748	1.95	117.95	130
average							249.4	1057.78

TABLE II

COMPARISON ON INDUSTRY BENCHMARKS

Circuit	REWIRE [12]		unreachable states [16]		Our method		Ratio on	Ratio on
	alt wires	CPU time (sec)	unreachable cube	CPU time (sec)	alt wires	CPU time (sec)	alt wires(%)	CPU time (%)
DESIGN1	638	0.13	172	0.63	2542	0.18	398.43	138.46
DESIGN2	741	0.42	295	5.60	749	0.55	101.08	130.95
DESIGN4	2463	1.49	3201	16.54	13955	2.35	566.59	157.72
DESIGN5	12662	8.7	36002	224.13	14937	22.35	117.97	256.9
DESIGN6	3336	7.56	609	38.09	3429	8.34	102.79	110.32
DESIGN7	6734	2.7	7372	151.70	6833	4.82	101.47	178.52
average							231.39	162.14

alternative wires locatable comparing with REWIRE.

VI. CONCLUSION

In this paper, an extended redundancy addition and removal algorithm based on unreachable states for sequential circuits is studied and experimented. We introduce the new notion of sequential alternative wires caused by unreachable states and utilize the idea for locating more alternative wires where some are only valid under the context of sequential machines.

Experimental results show that the number of alternative wires found is doubled on average. Clearly, this new approach is quite useful for the sequential benchmarks with more unreachable states. In the future, we would like to investigate further on the potential extending for more rewiring flexibility under the consideration of multiple unreachable states.

REFERENCES

[1] K. T. Cheng and L. A. Entrena, Multi-level logic optimization by redundancy addition and removal. In *Proc. EDAC*, pp. 373-377, 1993.

[2] L. A. Entrena and K. T. Cheng, Sequential logic optimization by redundancy addition and removal. In *Proc. ICCAD*, pp. 310-315, 1993.

[3] L. A. Entrena and K. T. Cheng, Combinational and sequential logic optimization by redundancy addition and removal. In Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions, vol. 14 7, pp. 909-916, July, 1995.

[4] S. C. Chang, M. Marek-Sadowska, and K. T. Cheng, Perturb and Simplify: Multilevel Boolean network optimizer. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.. 15 12, pp. 1494-1504 , Dec, 1996.

[5] S. C. Chang, van Ginneken, L.P.P.P., and M. Marek-Sadowska, Circuit optimization by rewiring. In IEEE Transactions on Computer, vol. 48 9, pp. 962-969, Sep, 1999 .

[6] Chih-Wei (Jim) Chang, and Malgorzata Marek-Sadowska, Who are the alternative wires in your neighborhood?. In Proc. GLVLSI-2001, pp. 103-108, Mar, 2001.

[7] D. I. Cheng, C. C. Lin, and M. Marek-Sadowska, Circuit partitioning with logic perturbation. In Proc, ICCAD, pp. 650-655, 1995.

[8] S. C. Chang, K. T. Cheng, N. S. Woo, and M. Marek-Sadowska, Layout driven logic synthesis for FPGAs. In Proc. DAC, pp. 308-313, June, 1994.

[9] S. C. Chang, K. T. Cheng, N. S. Woo, and M. Marek-Sadowska, Post-layout logic restructuring using alternative wires. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16 6, pp. 587-596, June, 1997.

[10] Catherine L. Zhou, Wai-Chung Tang, Wing-Hang Lo, and Yu-Liang Wu, How much can logic perturbation help from netlist to final routing for FPGAs. In Proc. DAC, pp. 922-927, June, 2007.

[11] C. W. Chang and M. Marek-Sadowska, Single-Pass redundancy addition and removal. In Proc. ICCAD, pp. 606-609, Nov, 2001.

[12] S. C. Chang, L. Ginneken, and M. Marek-Sadowska, Fast Boolean optimization by rewiring. In Proc. ICCAD, pp. 262-269, 1996.

[13] M. A. Iyer and M. Abramovici, FIRE: A fault-independent combinational redundancy identification algorithm. In IEEE Trans. VLSI Syst., vol. 4, pp. 295-301, June, 1996.

[14] Wu C.A. and Lin T. H. and Huang S. L. and Huang C.Y.R. SAT-controlled redundancy addition and removal: a novel circuit restructuring technique. In Proc. ASPDAC, pp. 191-196, Jan, 2009.

[15] Uwe Glaser and Kwang-Ting Cheng, Logic Optimization by An Improved Sequential Redundancy Addition and Removal Technique. In Proc. ASPDAC, pp. 235-240, 1995.

[16] Feng Yuan and Qiang Xu, On Systematic Illegal State Identification for Pseudo-Functional Testing. In Proc. DAC, pp. 702-707, July, 2009.

[17] Wing-Hang Lo, and Y. L. Wu, Improving Single-Pass Redundancy Addition and Removal with Inconsistent Assignments. In Proc. VLSI-DAT, pp. 175-178, April, 2006.

[18] Berkeley Logic Synthesis and Verification Group. ABC: A system for syquential synthesis and verification.. http://www.eecs.berkeley.edu/ alanmi/abc/