Logic Optimization with Considering Boolean Relations

Tung-Yuan Lee, Chia-Cheng Wu, Chia-Chun Lin, Yung-Chih Chen[§], Chun-Yao Wang
Department of Computer Science, National Tsing Hua University, Hsincu, Taiwan, R.O.C.
[§]Department of Computer Science and Engineering, Yuan Ze University, Chungli, Taiwan, R.O.C.

***Abstract*—Boolean Relation (BR) is a many-to-many mapping between two domains. Logic optimization considering BR can exploit the potential flexibility existed in logic networks to minimize the circuits. In this paper, we present a logic optimization approach considering BR. The approach identifies a proper sub-circuit and locally changes its functionality by solving the corresponding BR in the sub-circuit without altering the overall functionality of the circuit. We conducted experiments on a set of MCNC benchmarks that cannot be further optimized by *resyn2* script in ABC. The experimental results show that the node counts of these benchmarks can be further reduced. Additionally, when we apply our approach followed by the *resyn2* script repeatedly, we can obtain 6.11% improvements in average.**

I. Introduction

Logic optimization has been studied for many decades, either for two-level [5][10][11][15], or multi-level logic networks [16][17]. Having a completely optimized result in terms of node count is a challenging problem [8][14]. Hence, many heuristics have been proposed for dealing with this problem. Some previous work used Don't Cares, e.g., External Don't Care (XDC), Satisfiability Don't Care (SDC), or Observability Don't Care (ODC) in the networks to optimize the circuits [4][12][13].

XDCs are impossible input combinations to a gate or module explicitly specified by a design SPEC. For example, the patterns 1010 to 1111 (10 to 15 in decimal) from a binary-coded decimal (BCD) number module are the XDCs to the next module because they will not occur.

SDCs are impossible local input combinations of a gate in a Boolean network. Considering a 2-input AND gate, $y = x_1 \wedge x_2$, and a gate S driven by $x_1,\ x_2$, and y. $(y,\ x_1,\ x_2)$ = (0, 1, 1) is not a possible input pattern to S and thus is an SDC of gate S.

Considering a Boolean network, $y_1 = x_1 \wedge x_2 \wedge x_3$ and $y_2 = y_1 \wedge x_1$. Since y_1 cannot be observed at y_2 when $x_1 = 0$. Hence, $x_1 = 0$ is the ODC of y_1. As a result, y_1 can be minimized as $y_1 = (x_1 \wedge x_2 \wedge x_3) \vee (x_1' \wedge x_2 \wedge x_3) = x_2 \wedge x_3$ when considering its ODC.

In addition to DCs, Boolean Relation (BR) can be also used to optimize logic networks. Since a BR is a many-to-many mapping between two domains, it has to be transformed into a function such that it can be realized in logic networks.

This work is supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 106-2221-E-007-111-MY3, MOST 106-2221-E-155-056, MOST 103-2221-E-007-125-MY3

Fig. 1(a) is a BR between two domains, X and Y. The input patterns $x_1x_2 = \{01,\ 10\}$ are both mapped to two output patterns $y_1y_2 = \{00,\ 11\}$. To realize the logic network, we have to solve this BR. Fig. 1(b) and Fig. 1(c) are two different results after solving the BR. The logic functions with respect to the mappings in Fig. 1(b) and Fig. 1(c) are $\{y_1 = x_1 \wedge x_2,\ y_2 = x_1' \wedge x_2'\}$ and $\{y_1 = x_2,\ y_2 = x_1'\}$. Obviously, the function derived from Fig. 1(c) is more simplified.

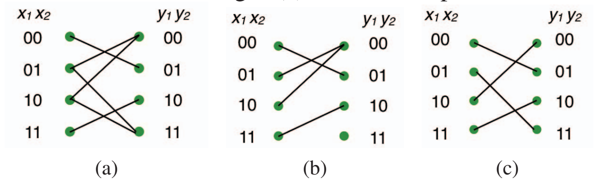


Fig. 1: (a) An example of Boolean Relation. (b)(c) Mappings after solving the BR.

In fact, how to solve BRs significantly influences the resultant functions. Several previous works proposed exact or heuristic optimizations to solve BRs under different cost functions, e.g., literal count [3][6][7][9][18] or BDD size [1].

In this paper, we propose an approach to optimize logic networks by exploiting BRs hidden in the network. First, we identify a many-to-one sub-circuit and a target sub-circuit divided by a cut as shown in Fig. 2(a) where the target sub-circuit is the circuit to be minimized. The many-to-one sub-circuit allows some input patterns to have the same output. Hence, we use these patterns to construct the hidden BRs in the target sub-circuit. Then we solve the BRs and obtain a more simplified target sub-circuit.

We conducted experiments on a set of MCNC benchmarks that cannot be further optimized by *resyn2* script in ABC [2]. The experimental results show that our approach can further reduce the number of nodes in the benchmarks. When we apply our approach followed by *resyn2* script repeatedly, we can reduce 6.11% nodes in average.

The main contributions of this work are two-fold:
(1) Our approach identifies the hidden BRs in the network for logic optimization.
(2) Our approach can reduce the node count of highly optimized circuits.

The rest of the paper is organized as follows. Section II gives an example of our idea. Section III introduces some backgrounds and definitions used in this paper. Section IV describes the proposed approach. Section V shows the experimental results and Section VI concludes this work.

II. A motivational example

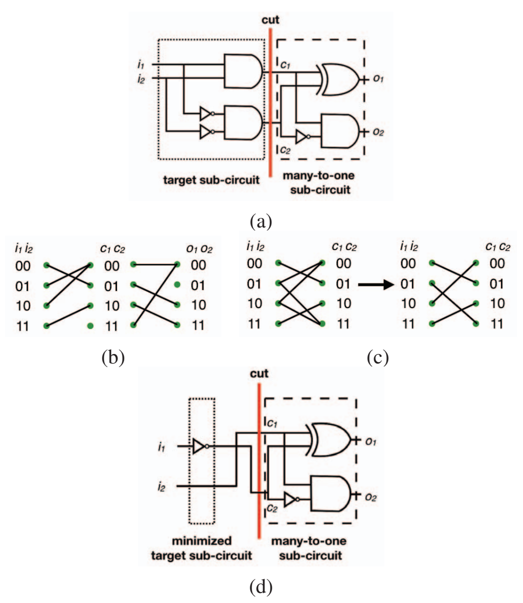


Fig. 2: (a) An original circuit. (b) The transition diagram. (c).The BR and its solution. (d) The optimized circuit.

In this section, we use an example in Fig. 2 to introduce our idea. Fig. 2(a) is an *optimization structure* where $i_1,\ i_2$ are the primary inputs (PIs), $c_1,\ c_2$ form a cut, and $o_1,\ o_2$ are the primary outputs (POs). In Fig. 2(a), we assume that the target sub-circuit (dotted square area) and many-to-one sub-circuit (dashed square area) have been identified.

Fig. 2(b) is the transition diagram of the circuit in Fig. 2(a). From Fig. 2(b), we observe that two patterns $\{00,\ 11\}$ at the cut produce the same output pattern $\{00\}$ at the POs, and two patterns $\{01,\ 10\}$ at the PIs produce $\{00\}$ at the cut. Hence, if we change the mapping of input patterns $\{01,\ 10\}$ from $\{00\}$ to $\{11\}$ at the cut, the overall functionality of the circuit is still intact. From this observation, we can construct a BR as shown in the left of Fig. 2(c). After solving the BR as shown in the right of Fig. 2(c), we can obtain simplified functions as $\{c_1 = i_2,\ c_2 = i_1'\}$. As a result, an optimized circuit with the minimized target sub-circuit is obtained as shown in Fig. 2(d) where two AND gates and one NOT gate in the original circuit of Fig. 2(a) are reduced.

III. Preliminaries

A. Boolean Relation

Boolean Relation (BR) is a many-to-many mapping between two domains. Let $B = \{0, 1\}$, a BR $R \subseteq X \times Y = B^n \times B^m$, where $X,\ Y$ are input and output domains with n and m dimensions.

A BR R is *well-defined* if and only if

$$\forall a \in B^n,\ \exists b \in B^m\ such\ that\ (a,\ b) \in R.$$

Since the derived BRs in this work come from Boolean networks, they are all well-defined BRs.

B. BR Solving

Solving a BR R is to seek a well-defined function f^1 such that $f \subseteq R$. We also say that f is a solution of R when we solve a BR to obtain f. Fig. 3(a) is a tabular representation showing the BR corresponding to Fig. 1(a). Fig. 3(b) is one solution of R. However, Fig. 3(c) is not a solution of R because $x_1x_2y_1y_2 = 1001$ (gray row) $\notin R$.

x_1x_2	y_1y_2
00	{01}
01	{00, 11}
10	{00, 11}
11	{10}

(a)

x_1x_2	y_1y_2
00	01
01	11
10	00
11	10

(b)

x_1x_2	y_1y_2
00	01
01	00
10	01
11	10

(c)

Fig. 3: (a) The BR R. (b) One solution of R. (c) Not a solution of R.

C. Many-to-one Boolean function

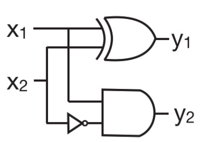
A Boolean function f with n input variables is *many-to-one* if and only if

$$\exists a_1, a_2 \in B^n,\ such\ that\ f(a_1) = f(a_2)\ where\ a_1 \neq a_2.$$

The circuit corresponding to a many-to-one Boolean function is a many-to-one circuit. Fig. 4(a) is the truth table of a many-to one function f, and Fig. 4(b) is the corresponding many-to-one circuit.

x_1x_2	y_1y_2
00	00
01	10
10	11
11	00

(a)



(b)

Fig. 4: (a) The truth table of a many-to-one Boolean function f. (b) The corresponding many-to-one circuit.

To group the input patterns that have the same output, we partition the input space into *compatible sets*. For a function f, the compatible set is a maximal subset S of input space such that $\forall a_i \in S,\ f(a_i)$ is identical. For example, the compatible sets of input space in Fig. 4(a) are $\{00,\ 11\}$, $\{01\}$, $\{10\}$ due to $f(00) = f(11)$. Note that the input space mentioned here is only corresponding to the considered sub-function.

IV. Algorithm

This section presents the proposed algorithm of our approach. It can be separated into three stages. First, we identify a many-to-one sub-circuit and generate its compatible sets. Then, we search for a target sub-circuit from the cut backward and build the corresponding BR. At last, we solve this BR to obtain a function, and re-synthesize the function for a more simplified Boolean network.

A. Extracting Flexibility

In the first stage, we identify a many-to-one sub-circuit and derive its compatible sets. We start with a node in the circuit, called starting node. Then, we gradually consider more gates connected to the starting node to form a many-to-one sub-circuit. The input of the many-to-one sub-circuit will become the cut of *optimization structure*. When the cut size is larger than a predefined value α, we terminate the process of many-to-one sub-circuit identification.

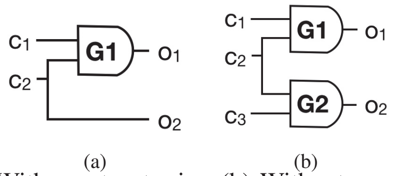


Fig. 5: (a) With a cut-out wire. (b) Without a cut-out wire.

The size of input space of a many-to-one sub-circuit is $2^{|cut|}$ and that of the output space is $2^{|output|}$. The average number of mapped input patterns at the cut for one output pattern is $\frac{2^{|cut|}}{2^{|output|}} = 2^{|cut|-|output|}$. Hence, the objective of identification algorithm is to maximize $(|cut| - |output|)$ such that a many-to-one sub-circuit can be identified with a higher probability.

For a many-to-one sub-circuit, some wires can be considered as cut wires and output wires simultaneously, e.g., $c_2,\ o_2$ in Fig. 5(a). These wires are called cut-out wires. The compatible set of a many-to-one sub-circuit will be influenced when a cut wire becomes a cut-out wire. For example, given a simple many-to-one sub-circuit with an AND gate, where $o_1 = c_1 \wedge c_2$, when c_1 and c_2 are cut wires solely, the compatible sets are {11}, {00, 01, 10}. However, if c_2 is a cut-out wire as shown in Fig. 5(a), c_2 also influences o_2 such that the compatible sets become {11}, {00, 10}, {01}.

We also use the example in Fig. 5 to demonstrate the process of many-to-one sub-circuit identification. At the beginning, the many-to-one sub-circuit is G1 only as shown in Fig. 5(a). $c_1,\ c_2$ form a cut and the compatible sets are {11}, {00, 10}, {01} for cut $c_1,\ c_2$ as mentioned.

When we further include gate G2 to form a many-to-one sub-circuit as shown in Fig. 5(b), the cut consists of $c_1,\ c_2,\ c_3$ and the compatible sets are {000, 001, 010, 100, 101}, {011}, {110}, {111}.

As shown in the last example, when we include one more gate into a many-to-one sub-circuit, we have to update the compatible sets as well. Here, we propose an efficient method instead of exhaustive simulation for this update. The update is categorized into two cases, Case A and Case B, based on whether the output of newly considered gate is at the original cut or not.

Case A: The output of newly considered gate is *not* at the original cut. For Case A, we further separate it into Case A1 and A2 based on whether the cut size increases or not. Fig. 6 illustrates two examples of Case A1 and A2. In these examples, the newly considered gate is y_2 (dotted line). The cut size of Case A1 increases while that of Case A2 does not. The many-to-one sub-circuit for Case A1 is shown in Fig. 6(a). The original compatible sets before updating are $x_1x_2 = \{00,\ 11\},\ \{01,\ 10\}$. Since x_3 is added as a new variable in the cut, the original compatible set {00, 11} is extended to {00-, 11-}. Then, we AND ($\cap$) this extended compatible set with the offset {-00, -01, -10} and onset {-11} of y_2, respectively, as shown in Fig. 6(b). That is, {00-, 11-} $\cap$ {-00, -01, -10} = {00-, 110} and {00-, 11-} $\cap$ {-11} = {111}. For the compatible set {01, 10}, the updating process is the same as shown in Fig. 6(b).

The many-to-one sub-circuit for Case A2 is shown in Fig. 6(c). Since the cut is intact after including y_2, the compatible set {00, 11} directly conducts AND operation with the offset {00, 01, 10} and onset {11} of y_2, respectively, for updating, as shown in Fig. 6(d).

Case B: The output of newly considered gate is at the cut. For Case B, we also separate it into Case B1 and B2 based on whether an input of newly considered gate is at the original cut or not. Fig. 7 illustrates two examples of Case B1 and Case B2. In these examples, the newly considered gate is at x_1. Any input of newly considered gate for Case B1 is *not* at the cut $x_1,\ x_2$ while that for Case B2 is at the cut $x_1,\ x_2$. The many-to-one sub-circuit for Case B1 is shown in Fig. 7(a). In Fig. 7(a), the inputs $x_3,\ x_4$ of newly considered gate x_1 are not at the original cut. Hence, we replace the value of x_1 with x_3 and x_4 during compatible set update. In the example of Fig. 7(b), for $x_1 = 0$, we replace x_1 with the offset of x_1, i.e., x_3x_4 = {00, 01, 10}. Then the compatible set x_1x_2 = {00} becomes $x_3x_4x_2$ = {000, 010, 100}. Similarly, for $x_1 = 1$, we replace x_1 with the onset of x_1, i.e., x_3x_4 = {11} and the compatible set x_1x_2 = {11} becomes $x_3x_4x_2$ = {111}.

The many-to-one sub-circuit for Case B2 is shown in Fig. 7(c). In Fig. 7(c), the input x_2 of the newly considered gate x_1 is overlapped with the original cut $x_1,\ x_2$. To update the compatible set, we first add a variable x_3 in the compatible sets as shown in the second column of Fig. 7(d). Since certain patterns in the extended compatible sets do not comply with the function of newly considered gate, AND gate, we remove these patterns from the compatible sets. For example, $x_1x_2x_3$ = {110} is removed out from the compatible set such that {11-} becomes {111} as shown in the third column of Fig. 7(d). Finally, we delete the variables not in the new cut, and the updated compatible sets are x_2x_3 = {0-}, {11}, {10} as shown in the last column of Fig. 7(d).

B. Building Boolean Relation

After deriving the compatible sets at the cut of optimization structure in the previous stage, we next identify a target sub-circuit and build the corresponding BR R. To identify a target sub-circuit, we apply a Breadth-First Search (BFS) method in the Boolean network from the cut backward. Since the gates in the target sub-circuit could be removed after optimization, they can only connect to nodes in the target sub-circuit itself or wires at the cut. The BFS procedure will stop at the PIs, the POs, or the wires connecting to gates not in the target sub-circuit. These stopping points then become the inputs of the target sub-circuit.

Fig. 8 illustrates an example of identifying a target sub-circuit from the cut $c_1,\ c_2$. Fig. 8(a) is a Boolean network to be explored, where $c_1,\ c_2$ form the cut, i_1 is a PO, and i_2 is a PI. First, the gate c_1 is selected as shown in Fig. 8(b). Then, the gate c_2 is selected based on the BFS method as shown in Fig. 8(c). Finally, the gate x_1 is selected because it only connects to the target sub-circuit itself as shown in Fig. 8(d).

[1]A function f is well-defined if and only if $\forall a \in B^n,\ \exists b \in B^m$ *such that* $(a,\ b) \in f$.

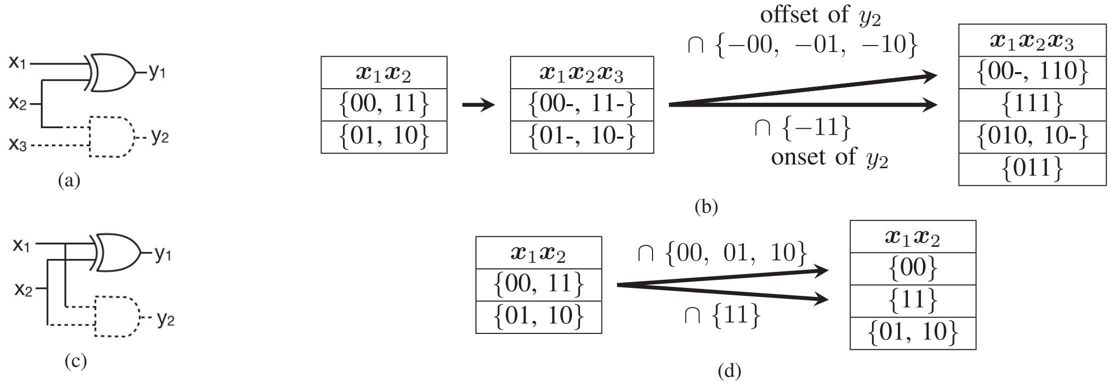


Fig. 6: (a) The sub-circuit for Case A1. (b) The update of compatible sets for Case A1. (c) The sub-circuit for Case A2. (d) The update of compatible sets for Case A2.

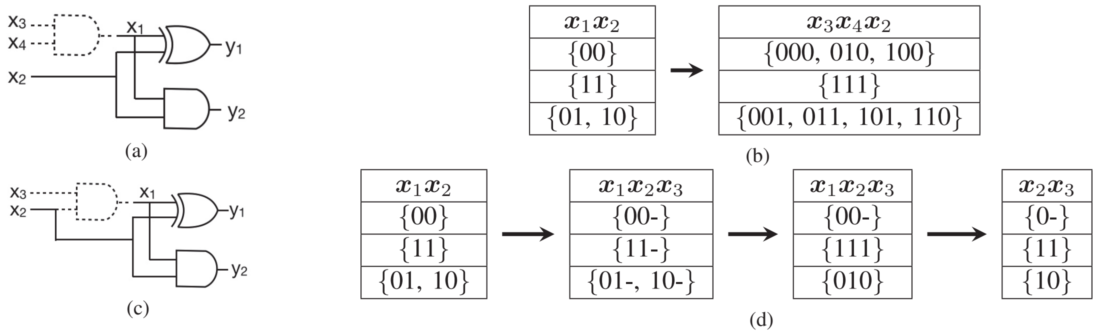


Fig. 7: (a) The sub-circuit for Case B1. (b) The update of compatible sets for Case B1. (c) The sub-circuit for Case B2. (d) The update of compatible sets for Case B2.

The identification procedure is stopped at i_1, i_2, and i_3 since i_1 is a PO, i_2 is a PI, and i_3 is connected to one inverter that does not belong to the target sub-circuit itself.

After identifying the target sub-circuit as shown in Fig. 8(d), we derive the truth table for the target sub-circuit as shown in Fig. 8(e). With the truth table of target sub-circuit and the compatible set from the many-to-one sub-circuit, we can build a BR of the target sub-circuit.

Next, the SDCs, i.e., the illegal input combinations among the inputs of target sub-circuit, are further computed. Considering this SDC information will relax the BR such that a more simplified function could be obtained easily. To compute SDCs, we also apply a BFS method from each input of target sub-circuit backward to get a corresponding transitive fanin cone (TFIC) such as TFIC$_{i_1}$ and TFIC$_{i_3}$ as shown in Fig. 8(f)[2].

When there exist other inputs of the target sub-circuit in the TFIC of one input, SDCs may occur at the target sub-circuit. For example, since i_2 appears in the TFIC of i_3, TFIC$_{i_3}$, as shown in Fig. 8(f), SDCs may exist. Next, we apply full simulations for TFIC$_{i_3}$ and realize that $i_2i_3 = 11$ never occurs due to TFIC$_{i_3}$ = NOR gate. Hence, $i_1i_2i_3 = -11$ are the SDCs for the target sub-circuit. To reduce the complexity of simulation, the number of inputs for each clustered TFIC is limited to a predefined value β.

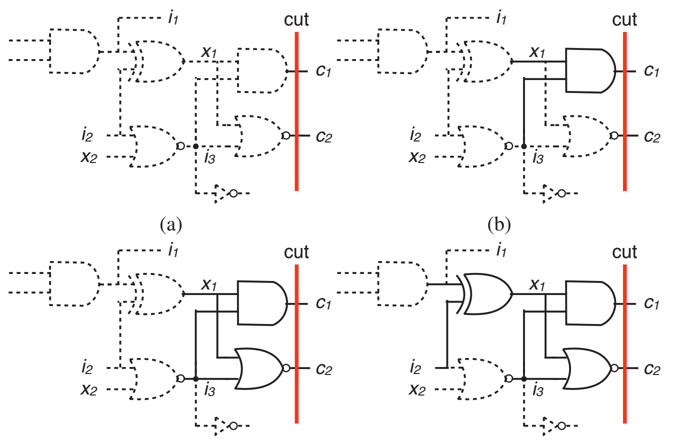
Using the compatible set {00, 11}, {01}, {10} from the many-to-one function in Fig. 4(a), the truth table in Fig. 8(e), and the computed SDCs $i_1i_2i_3 = $ -11 for the target sub-circuit, we can build a corresponding BR for the target sub-circuit. Since {00, 11} is a compatible set, we can replace c_1c_2 = 00 in Fig. 8(e) with {00, 11}as shown in the gray rows. We also replace c_1c_2 with {--} for input patterns in the SDCs $i_1i_2i_3 = 011$ and 111 as shown in the dark gray rows. Fig. 8(g) is the resultant BR for the target sub-circuit after the replacements.

[2]Since i_2 is a PI, TFIC$_{i_2}$ is empty.

C. Solving Boolean Relation

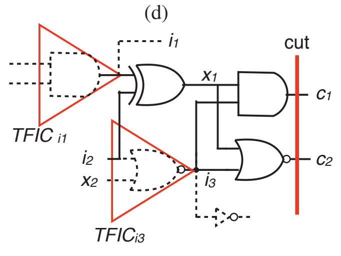
Solving a BR influences the target sub-circuit minimization. In this stage, we customize a divide-and-conquer based method proposed in [1] to solve BRs.

The cost function of our approach is the gate count after synthesis. We use the same BR as shown in Fig. 8(g) to introduce the method for solving BRs. Note that the objective of solving BRs is to obtain a smaller target sub-circuit. The original target sub-circuit is shown in Fig. 8(f) with three gates. Hence, the upper bound of cost is three. Given the BR R as shown in Fig. 8(g) where the inputs are i_1, i_2, and i_3 and the outputs are c_1 and c_2, to solve R, we first project R onto each output c_1 and c_2 independently (denoted as $R \downarrow c_i$). The Karnaugh-maps (K-maps) for $R \downarrow c_1$ and $R \downarrow c_2$ are shown in Fig. 9(a). When both 0 and 1 appear in c_i for one input pattern, the corresponding K-map is filled with don't care "-". In this example, $c_1 = i_3$ and $c_2 = i_3'$ after minimizing the functions in Fig. 9(a). Since the cost is only one, which is smaller than three, we further concatenate c_1 and c_2 to form a multiple-output function as shown in Fig. 9(b). Unfortunately, this function is not a correct solution of R. For example, when



$i_1i_2i_3$	c_1c_2
000	01
001	00
010	00
011	10
100	00
101	10
110	01
111	00

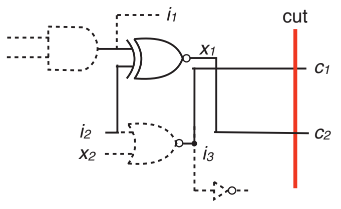
(e)



(f)

$i_1i_2i_3$	c_1c_2
000	{01}
001	{00, 11}
010	{00, 11}
011	{--}
100	{00, 11}
101	{10}
110	{01}
111	{--}

(g)



(h)

Fig. 8: An example of identifying a target sub-circuit and build the BR of it.

$i_1 \backslash i_2i_3$	00	01	11	10
0	0	-	-	-
1	-	1	-	0

$c_1 = i_3$

$i_1 \backslash i_2i_3$	00	01	11	10
0	1	-	-	-
1	-	0	-	1

$c_2 = i_3'$

(a)

$i_1i_2i_3$	c_1c_2
000	01
001	10
010	01
011	10
100	01
101	10
110	01
111	10

(b)

Fig. 9: (a) The K-maps for $R \downarrow c_1$ and $R \downarrow c_2$. (b) The truth table of the solution of R with three conflicts.

$i_1i_2i_3 = 001$, it should map to $c_1c_2 = 00$ or 11 instead of 10 according to the BR. We call this situation a conflict. Thus, the original BR is split into two BRs, R_1 and R_2, from a conflict input as shown in Fig. 10. Then we repeatedly apply the same method to solve R_1 and R_2.

After solving R_1, we obtain $c_1 = i_3$ and $c_2 = (i_1 \oplus i_2)'$ without any conflicts. Since the corresponding sub-circuit has only one XNOR gate and the cost is one, we accept this solution[3] and the upper bound of cost is updated as one. When solving a BR, if the resultant cost is larger than the current upper bound, we discard the BR without checking conflicts.

$i_1i_2i_3$	c_1c_2
000	{01}
001	{00, 11}
010	{00, 11}
011	{--}
100	{00, 11}
101	{10}
110	{01}
111	{--}

(a)

R_1

$i_1i_2i_3$	c_1c_2
000	{01}
001	{11}
010	{00, 11}
011	{--}
100	{00, 11}
101	{10}
110	{01}
111	{--}

(b)

R_2

$i_1i_2i_3$	c_1c_2
000	{01}
001	{00}
010	{00, 11}
011	{--}
100	{00, 11}
101	{10}
110	{01}
111	{--}

(c)

Fig. 10: (a) The original BR R. (b)(c) R_1, R_2 after splitting.

Then we solve R_2 and obtain $c_1 = (i_2 \wedge i_3') \oplus i_1$ and $c_2 = i_3'$. Since the cost of this solution is larger than the current upper bound of cost, we discard this solution as mentioned and terminate this stage. The minimized target sub-circuit is shown in Fig. 8(h).

The flow chart of the proposed approach is shown in Fig. 11. First, we extract flexibility from a given starting node by constructing the many-to-one sub-circuit. Then, we identify the target sub-circuit and build the corresponding BR considering SDCs. At last, we solve the BR to obtain a function, and synthesize it for a more simplified target sub-circuit. We repeat these stages for considering each node as a starting node to minimize the whole Boolean network.

V. Experimental Results

The proposed approach was implemented in C++. The experiments were performed on a Linux CentOS 6.7 workstation with Intel(R) Xeon(R) E5-2650V2 CPU @ 2.60GHz 64GB RAM for a set of MCNC benchmarks [19] represented by And-Inverter-Graph (AIG) [2]. The parameters for cut size limit (α) is 6, and for TFIC input limit (β) is 15 in the experiments.

To demonstrate the ability of exploring optimization potential of our approach, each benchmark in the experiment was highly optimized by applying $resyn2$ script in ABC [2] repeatedly until the circuit is intact. We conducted two experiments: One is running our approach once only; the other is combining our approach with $resyn2$ script together.

Table I summarizes the experimental results. Column 1 lists the benchmark information including name, the number of PIs, POs and node count. Column 2 shows the number of reduced nodes, the percentage of reduced nodes, and the required CPU time measured in second when running our approach once only.

Column 3 shows the corresponding results of the second experiment. In this experiment, we conducted our approach followed by $resyn2$ script and repeated the iterations until

[3]We also accept a solution when its cost is equal to the current upper bound. This is because restructuring a Boolean network could bring new opportunities for further optimization.

TABLE I: Experimental results of our approach.

Circuit	PI	PO	|node|	Ours			(Ours+resyn2)×k			
				n_r	n_r(%)	CPU(s)	n_r	n_r(%)	k	CPU(s)
alu2	10	6	353	2	0.57	217.21	5	1.42	2	378.33
apex6	135	99	604	1	0.17	118.74	6	0.99	2	242.36
i10	**257**	**224**	**1695**	**4**	**0.24**	**460.51**	**69**	**4.07**	**13**	**6284.58**
i9	88	63	522	3	0.57	163.12	12	2.30	2	363.47
pair	173	137	1261	1	0.08	328.12	28	2.22	4	1132.28
term1	34	10	138	2	1.45	117.49	6	4.35	1	117.49
dalu	75	16	1052	2	0.19	189.19	23	2.19	5	891.61
cordic	23	2	270	1	0.37	114.77	73	27.04	6	453.84
i8	133	81	954	4	0.42	166.55	35	3.67	2	355.14
apex7	49	37	173	4	2.31	37.94	7	4.05	1	37.94
cmb	16	4	47	4	8.51	14.13	4	8.51	1	14.13
x1	51	35	545	2	0.37	171.82	68	12.48	6	948.47
Average	-	-	-	2.5	1.27	174.97	28	6.11	-	934.97

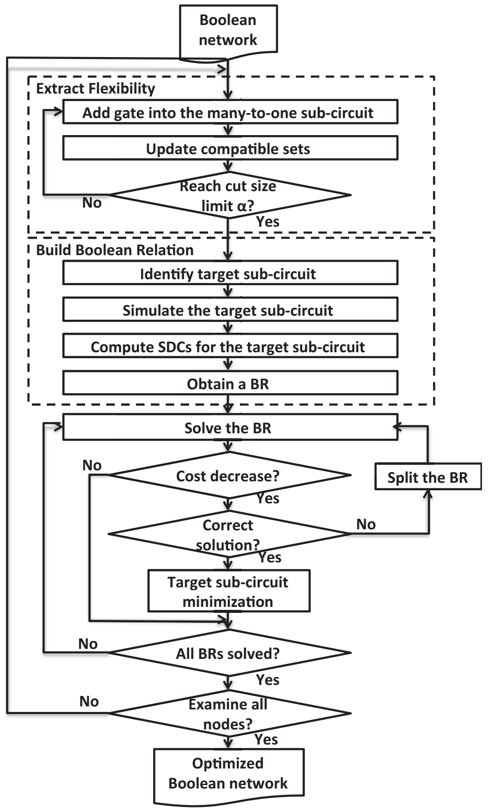


Fig. 11: The flow chart of the proposed approach.

the circuit is intact. The k value in the table is the iteration number for each benchmark and it varies for different circuits.

For example, for *i10* benchmark, our approach cost 460 seconds to reduce four nodes. However, when we conducted our approach with *resyn2* repeatedly, 69 AIG nodes, or 4.07%, can be reduced after 13 iterations.

According to Table I, we observed that for these highly optimized benchmarks, our approach still can reduce the node count, but not many. In fact, we admit that for some benchmarks, the proposed approach does not work very well due to highly optimized nature of the benchmark by the excellent optimization script *resyn2*. However, by exploiting hidden BRs to perturb the sub-circuits, the resultant network could be further optimized by *resyn2* script. The node count reduction is up to 27.04%, and has an average of 6.11%.

VI. Conclusion

This paper proposes a logic optimization approach considering Boolean Relation hidden in the logic networks. To exploit the hidden BR, we first extract flexibility. Next, we identify a proper target sub-circuit and build the BR. At last, we solve the BR to optimize the target sub-circuit. The experimental results show that our approach can still reduce node count for a highly optimized circuit. When combining withe the *resyn2* script, we can obtain up to 27.04% reduction or 6.11% improvements in average.

References

[1] D. Baneres, J. Cortadella, and M. kishinevsky, "A recursive paradigm to solve boolean relations," in *Proc. DAC*, pp. 416-421, June 2004.

[2] Berkeley Logic Synthesis and Verification Group, ABC: a system for sequential synthesis and verification, Available: https://people.eecs.berkeley.edu/∼alanmi/abc/.

[3] R. Brayton and F. Somenzi. "An exact minimizer for boolean relations," in *Proc. ICCAD*, pp. 316-319, 1989.

[4] Y. C. Chen, and C. Y. Wang, "Fast node merging with dont cares using logic implications," in *Proc. TCAD*, vol. 29, no. 11, pp. 18271832, Nov. 2010

[5] O. Coudert, J. Madre, and H. Fraisse, "A new viewpoint on two-level logic minimization," in *Proc. DAC*, pp. 625-630, June 1993.

[6] A. Ghosh, S. Devadas, and A. Newton, "Heuristic minimization of boolean relations using testing techniques." in *Proc. TCAD*, 1990.

[7] S. Jeong, and F. Somenzi, "A new algorithm for the binate covering problem and its application to the minimization of boolean relations," in *Proc. ICCAD*, pp. 417-420, 1992.

[8] E. L. Lawler, "An approach to multilevel boolean minimization," Journal of the ACM, 1964.

[9] B. Lin and F. Somenzi. "Minimization of symbolic relations," in *Proc. ICCAD*, pp. 88-91, 1990.

[10] E. J. McCluskey, "Minimization of boolean functions," *Bell Syst. tech J.*, vol. 35, no. 5, pp. 1417-1444, Nov. 1956.

[11] P. McGeer, J. Sanghavi, and R. K. Brayton, "Espresso-signature: A new exact minimizer for logic functions," in *Proc. TVLSI*, pp. 618-624, 1993.

[12] A. Mishchenko, and R. K. Brayton, "Simplification of non-deterministic multi-valued networks," in *Proc. ICCAD*, November, 2002.

[13] A. Mishchenko, and R. K. Brayton, "A theory of non-deterministic networks," in *Proc. TCAD*, June, 2006.

[14] C. D. Murray, and R. R. Williams, "On the (non) NP-Hardness of Computing Circuit Complexity," in *Theory of Computer*, June, 2017.

[15] W.Quine, "The problem of simplifying truth functions," American Mathematical Monthly, vol. 59, no. 8, pp. 521-531, 1952.

[16] H. Savoj, "Improvements in technology independent optimization of logic circuits," *Proc. of IWLS'97*.

[17] H. Savoj, and R. K. Brayton, "The use of observability and external don't-care for the simplification of multi-level networks," *Proc. DAC*, pp. 291-301, 1990.

[18] Y.Watanabe, and R. K. Brayton, "Heuristic minimization of multi-valued relations," in *Proc. TCAD*, pp. 1458-1472, Oct. 1993.

[19] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0," Microelectronics Center of North Carolina, 1991.