Using Structural Relations for Checking Combinationality of Cyclic Circuits

Wan-Chen Weng, Yung-Chih Chen[§], Jui-Hung Chen, Ching-Yi Huang, Chun-Yao Wang
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[§]Department of Computer Science and Engineering, Yuan Ze University, Chungli, Taiwan, R.O.C.

***Abstract*—Functionality and combinationality are two main issues that have to be dealt with in cyclic combinational circuits, which are combinational circuits containing loops. Cyclic circuits are combinational if nodes within the circuits have definite values under all input assignments. For a cyclified circuit, we have to check whether it is combinational or not. Thus, this paper proposes an efficient two-stage algorithm to verify the combinationality of cyclic circuits. A set of cyclified IWLS 2005 benchmarks are performed to demonstrate the efficiency of the proposed algorithm. Compared to the state-of-the-art algorithm, our approach has a speedup of about 4000 times on average.**

I. Introduction

Digital circuits can be classified into two categories, sequential circuits and combinational circuits. Unlike sequential circuits, combinational circuits are generally designed without loops. However, previous works [12] [17] have shown that to reduce areas of combinational circuits, designers should not restrict the structures of combinational circuits to loop-free topologies. Cyclic combinational circuits are combinational circuits, even though they have loops. After introducing loops into combinational circuits, if the outputs are still *definite* under *every* input assignment, the loops are *false*. Otherwise, the outputs are indefinite under some input assignments and the loops are *true*. For example in Fig. 1(a), implementing these seven combinational functions needs 9 gates. After introducing a loop as shown in Fig. 1(b), however, the gate count is reduced to 7. Additionally, each output function of the circuit remains unchanged and the combinationality of the circuit in Fig. 1(b) is also preserved. This is because the introduced loop is false actually. We can see that when the input a is 0 and 1, the loop is broken in gate $n5$ and gate $n9$, respectively. This implies that the outputs are definite for all input assignments, i.e., (a, b, c, d, e) = (0, −, −, −, −) and (1, −, −, −, −) where "−" means don't-care.

Aside from the area minimization, cyclification provides a timing benefit as well. [16] has shown that cyclic circuits can not only remove logic gates off the critical path but also shorten the circuit delay. For instance in Fig. 1(a), assume that the gate delay is 1 and wire delay is 0 for simplicity. The numbers in the dotted circles are the delay at the output of each gate. We can see that the circuit delay is 8 with respect to a path from gate $n1$ to $n8$ when the input assignment (a, b, c, d, e) = (1, 0, 1, 0, 0). Under this input assignment, the side-inputs of the path are all input-noncontrolling values. Thus, this path is sensitized and is a critical path of the acyclic circuit. However, in Fig. 1(b), the critical path delay can be reduced to 7 for the cyclic version of Fig. 1(a).

Although cyclic combinational circuits could have area and timing advantages, most previous works as in [9] [13] intended to construct an acyclic alternative to a cyclic circuit so that traditional analysis methods for combinational circuits could be applied effectively. Riedel and Bruck [14] [15] worked on the synthesis of cyclic combinational circuits. Additionally, due to the potential memory explosion with the *binary decision diagram* (BDD)-based algorithm [14] [15], Backes et al. [1] [2] [3] exploited the Craig interpolation [8] to synthesize cyclic combinational circuits and developed a *satisfiability* (SAT)-based approach to verify the combinationality of cyclic circuits by using a dual-rail model. With the dual-rail model, the validation time in [1] was large since each signal in loops was duplicated and the problem size increases rapidly. Consequently, as the circuit size grows, the approach becomes inefficient.

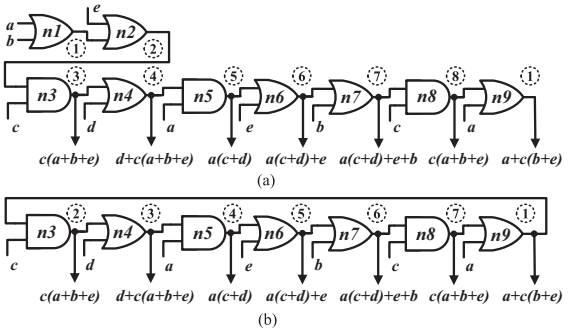


Fig. 1. An example for further improvement in area and timing by introducing combinational loops. (a) Original implementation. (b) Cyclic implementation.

Thus, in this paper, we propose a two-stage algorithm for checking a cyclic circuit's combinationality. The first stage is to identify combinational circuits efficiently using a conflict analysis method. For circuits that cannot be identified as combinational by the conflict analysis method, the second stage exploits backtracking method and SAT-solver to determine the combinationality of the circuits. Experimental results on a set of cyclified IWLS 2005 benchmarks show that the CPU time is still little as the circuit size grows, demonstrating the scalability and efficiency of our algorithm.

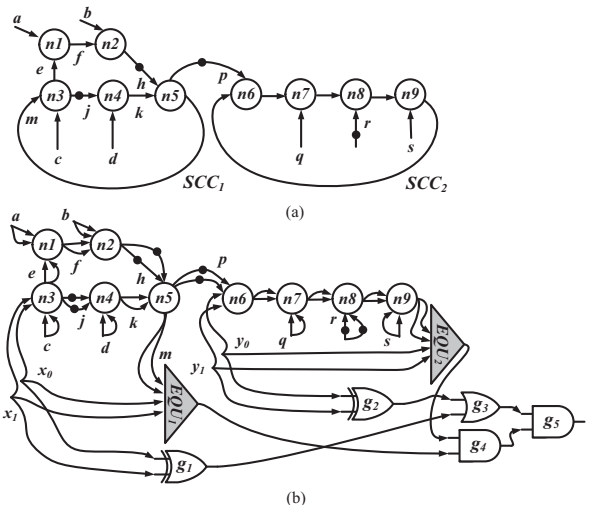
The main contributions of this work are two-fold: 1. We propose a conflict analysis method that efficiently identifies combinational cyclic circuits. 2. We propose a backtracking method that transforms the problem of combinationality checking to a SAT problem with a more efficient model.

II. Preliminaries

In this section, we introduce some previous works and the background of this work. [6] [7] proposed an algorithm to detect nodes that can be merged in a Boolean circuit without changing functionality. By replacing a target node n_t with a substitute node n_s where n_s is in the transitive fanout cone of n_t, a cyclic structure will be introduced. Since cyclic combinational circuits could achieve a more minimized area, [5] introduced loops into the circuits. However, the circuit's combinationality has to be verified for achieving this objective.

A pair of nodes $n1$ and $n2$ are said strongly connected if there exist paths between this pair of nodes in both directions. In a directed graph G, if every pair of nodes within a subgraph S of G is strongly connected, S is a *strongly connected component* (SCC) of G. The fanins to S which cannot be reached from the nodes of S are the *side-inputs* of S. An SCC was used as a basic component for representing a set of loops [9]. Once all SCCs in a circuit pass the validation procedure, the combinationality of the circuit is guaranteed. Thus, in the following discussion, for simplicity, we only focus on the SCC which represents a cyclic sub-circuit of the original circuit.

This work is supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 103-2221-E-007-125-MY3, MOST 103-2221-E-155-069, NSC 102-2221-E-007-140-MY3, and NSC 100-2628-E-007-031-MY3.



(a)

(b)

Fig. 2. (a) An example of cyclic circuit containing two SCCs. (b) The dual-rail model of Fig. 2(a).

[1] proposed a SAT-based approach using a dual-rail model to check the combinationality of an SCC with a ternary framework, $\{0, 1, \perp\}$. The ternary framework is used to verify if the internal signals would be indefinite, denoted as $\perp$. The notation $\perp$ represents an uncertain value. Additionally, to analyze cyclic circuits with the set of ternary values, the authors first find the cutting points that will break all the loops in the SCC. Then, they add dummy variables at the cutting points and duplicate internal signals with the encoding scheme: 1. two-bit (1, 1) represents logical 1. 2. two-bit (0, 0) represents logical 0. 3. (0, 1) and (1, 0) represent $\perp$. For each dummy variable, which is encoded in two bits, an equivalence checker (EQU) is set up to see if the values assigned to dummy variables agree with that computed by the circuit at the cutting points. Furthermore, to recognize $\perp$ within a loop, an XOR gate is added to check if cyclic portion of the circuit will generate or propagate $\perp$ under some input assignments. If any XOR gate detects $\perp$ and the results of all EQUs are "equivalent", indefinite values exist in its original cyclic circuit, and this cyclic circuit is non-combinational.

Although the approach using the dual-rail model can successfully verify the combinationality of a circuit, it is not scalable. For example, a cyclic circuit represented in *And-Inverter Graph* (AIG) is shown in Fig. 2(a), where a node represents an AND gate, and a dot represents an INV gate. Fig. 2(a) contains 9 nodes, 3 loops, and 18 signal variables. Its corresponding dual-rail model as shown in Fig. 2(b), however, has 4 dummy variables x_0, x_1, y_0, and y_1, and 55 signal variables due to the signal duplication. Thus, as the size of circuit grows, the corresponding *conjunctive normal form* (CNF) of the dual-rail model becomes larger and might cause the SAT-solving time increase significantly.

III. The Algorithm

In this section, we present the proposed two-stage algorithm for verifying the combinationality of an SCC.

A. Combinationality

In the beginning, we present the theoretical background about combinationality of an SCC. In [9], a theorem about the combinational behavior of cyclic circuits has been proposed. The theorem states that if a circuit containing an SCC behaves combinationally, at least one side-input to the SCC is a controlling value. However, if there exists a controlling side-input to an SCC, the SCC could be still non-combinational. Take SCC_1 in Fig. 2(a) as an example where a, b, c, and d are side inputs of SCC_1. When a is assigned as 0, a is a controlling side-input for SCC_1 since 0 is the controlling value for AND gate $n1$. After performing logic implication with (a, b, c, d) = (0, 1, 1, 1) as shown in Fig. 2(a), some nodes in the loop $n1 - n2 - n5 - n3$ still have $\perp$ and the loop is a true loop. Therefore, SCC_1 is still non-combinational even it has a controlling side-input a.

We observe that side-inputs to *loops* in an SCC is more related to the combinationality of the SCC. Thus, we first investigate the relationships between side-inputs of loops in an SCC and SCC's combinationality.

Definition 1. Given a loop L with indefinite ($\perp$) node values, we say side-inputs I of L *dominate* L if and only if there is a subset of I controlling the node values of L when applying each input assignment to the nodes of L.

In summary, an SCC being combinational is equivalent to every loop in the SCC being dominated by one controlling side-input. In other words, if there exists one loop whose side-inputs are all non-controlling values for an input assignment, the SCC is non-combinational. We will use this proposition when exploiting SAT-solvers to determine the combinationality of a loop in Section III-C.

B. Self-conflict Detection

According to *Theorem 1*, side-inputs of loops in an SCC play an important role in validating the SCC's combinationality. We classify the side-inputs of loops in an SCC into two types: *Outer side-inputs* (OSIs): those from the outside of an SCC; *Inner side-inputs* (ISIs): those from other loops that belong to the same SCC. For example in Fig. 2(a), the OSIs of loop $n3 - n4 - n5$ are c and d, and the ISI is $\overline{n2}$.

We observe that if the functions of two side-inputs of a loop are complemented in an AIG, the loop can be detected as false directly. We name these two side-inputs as *self-conflict side-inputs* (SCSIs). For example in Fig. 2(a), if the functions of side-inputs r and q of the loop $n6 - n7 - n8 - n9$ are complemented, $r = \overline{q}$, the loop is false and SCC_2 is combinational. This is because according to *Theorem 1*, all the assignments to the side-inputs of the loop (p, q, r, s) = ($-$, 0, 1, $-$) and ($-$, 1, 0, $-$) have one controlling side-input dominating the loop. Thus, SCC_2 is detected as combinational.

Note that only OSIs of a loop are examined for SCSI detection. ISIs are excluded from this detection because they are not always definite. Furthermore, if OSIs are within the fanout cones of other SCCs, they also possibly have indefinite values. Thus, only OSIs that do not have loops in their fanin cones are considered for self-conflict detection.

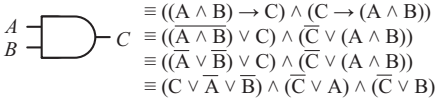
C. Loop Backtracking

If there do not exist SCSIs in a loop, we have to further examine if there exists an input assignment such that all side-inputs of a loop are non-controlling values. If no such an input assignment exists (UNSAT), the loop is false and the SCC is combinational. Here we propose a backtracking method with SAT-solver to deal with this problem. In the proposed method, we build CNF formulae for the side-input functions of each loop in an SCC, then we use SAT-solver to examine whether an input assignment causing all side-inputs of the loop be non-controlling values exists or not.

Note that the CNF construction for loops is a little different from that for acyclic circuits. If we construct the CNF of cyclic circuits through traditional CNF construction algorithms that are for acyclic circuits such as Tseitin [18], the results returned by SAT-solver might be erroneous or misleading. This is because loops do not have ending points, and oscillation happened in node values of a loop could be neglected by SAT-solver. For example in SCC_1 of Fig. 2(a), assume that $k = 0$ and other internal nodes are indefinite initially when the OSIs a, b, c, and d are all non-controlling values, 1, to

the nodes in SCC_1. Then we have $n5 = 0$, $n3 = 0$, $j = 1$, and $k = 1$. We find that k is changed from 0 to 1. After $h = 1$ through the path $n3 - n1 - n2 - h$, $n5$ is also changed from 0 to 1. As a result, the value oscillation occurs in a loop and the circuit is non-combinational. Note that in the SAT-solving process, if a node has been assigned conflict values, which means no assignment exists, SAT-solving result will be UNSAT. However, UNSAT is considered as a result for combinational circuits in our proposed algorithm. Therefore, we have to modify the method of CNF construction for cyclic circuits. We will discuss this in the following paragraphs.

Fig. 3 shows the CNF construction for an AND gate through [18] where A and B are the inputs and C is the output. For an OSI that is not within the fanout cones of other SCCs, we directly apply Tseitin [18] transformation to construct the CNF of the side-input function until reaching the PIs. For other OSIs or ISIs, however, we have to modify the construction method due to node re-visit during the loop backtracking. The proposed modification method for the CNF construction is to change the node names when nodes are re-visited and treat these nodes as virtual PIs. The loop backtracking process will be terminated on these virtual PIs as well.



$$
\begin{aligned}
&(\mathrm{A} \wedge \mathrm{B}) \leftrightarrow \mathrm{C} \\
&\equiv ((\mathrm{A} \wedge \mathrm{B}) \rightarrow \mathrm{C}) \wedge (\mathrm{C} \rightarrow (\mathrm{A} \wedge \mathrm{B})) \\
&\equiv ((\overline{\mathrm{A} \wedge \mathrm{B}}) \vee \mathrm{C}) \wedge (\overline{\mathrm{C}} \vee (\mathrm{A} \wedge \mathrm{B})) \\
&\equiv ((\overline{\mathrm{A}} \vee \overline{\mathrm{B}}) \vee \mathrm{C}) \wedge (\overline{\mathrm{C}} \vee (\mathrm{A} \wedge \mathrm{B})) \\
&\equiv (\mathrm{C} \vee \overline{\mathrm{A}} \vee \overline{\mathrm{B}}) \wedge (\overline{\mathrm{C}} \vee \mathrm{A}) \wedge (\overline{\mathrm{C}} \vee \mathrm{B})
\end{aligned}
$$

Fig. 3. The CNF construction for an AND gate through Tseitin [18].

For example, the process of backtracking from $n4$ in Fig. 2(a) is shown in Fig. 4. When $n4$ and $n3$ are re-visited, they are renamed as $n4^*$ and $n3^*$, respectively. $n4^*$ and $n3^*$ are virtual PIs and used to terminate this backtracking process. We also exploit conflict between the values in ni and ni^*, if any, to detect oscillation.

We use an example represented in AIG as shown in Fig. 5(a) to demonstrate the CNF construction for the loop $n6 - n7 - n8 - n9$. Note that for the circuits that are not represented as AIGs, the proposed method is still appliable. Let us explain how to construct the CNF briefly. Our objective is to see if all the side-inputs of the loop can be assigned as non-controlling value, 1, simultaneously. Therefore, the side-inputs are ANDed together and the CNF is constructed as $(w \bigvee \overline{n5} \bigvee \overline{n10} \bigvee \overline{r} \bigvee \overline{s}) \bigwedge (\overline{w} \bigvee n5) \bigwedge (\overline{w} \bigvee n10) \bigwedge (\overline{w} \bigvee r) \bigwedge (\overline{w} \bigvee s)$ as mentioned in Fig. 3. After setting w as 1, the CNF can be simplified as $(n5 \bigwedge n10 \bigwedge r \bigwedge s)$, as shown in the first clause of Fig. 5(c). Then the backtracking process is performed recursively until reaching the PIs or virtual PIs as shown in Fig. 5(b). Note that $n11$ will also be reached twice since $n11$ belongs to the fanin cones of both $n5$ and $n10$. However, since there is no loop in the fanin cone of $n11$, the value of $n11$ is definite. Thus, no virtual PI is necessary for substituting $n11$ when $n11$ has been reached again from $n10$. The complete CNF is shown in Fig. 5(c). In Fig. 5(b), when we assign $n5$ and $n10$ as 1 simultaneously, there occurs a value conflict on $n11$. Therefore, the SAT-solver will return UNSAT indicating that no input assignment can make all the side-inputs of the loop non-controlling values. As a result, the loop $n6 - n7 - n8 - n9$ is combinational.

Fig. 6 shows the procedure of *Loop Backtracking*. In a loop L, every side-input i of L has to be backtracked. In the backtracking of a side-input i, if i has been backtracked from other loops, we ignore it; otherwise, we backtrack it. For each fanin f of i, if f has been reached before, a virtual PI f^* is used for substituting f. After backtracking each side-input of a loop, the clauses of each side-input are ANDed together to see whether all side-inputs can be non-controlling values simultaneously under an input assignment.

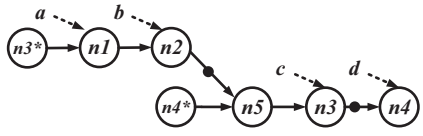
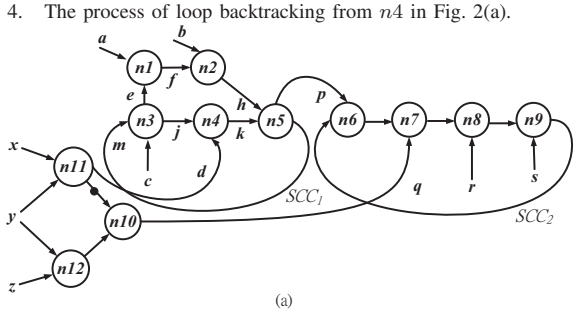
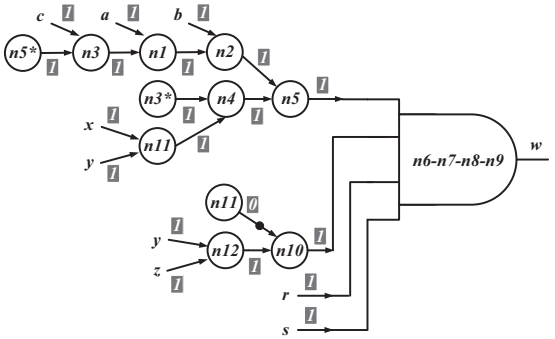


Fig. 4. The process of loop backtracking from $n4$ in Fig. 2(a).



(a)



(b)

$$
\begin{aligned}
&(n5 \wedge n10 \wedge r \wedge s) \wedge \\
&\left.\begin{array}{l}
(n5 \vee \overline{n4} \vee \overline{n2}) \wedge (\overline{n5} \vee n4) \wedge (\overline{n5} \vee n2) \wedge \\
(n2 \vee \overline{n1} \vee \overline{b}) \wedge (\overline{n2} \vee n1) \wedge (\overline{n2} \vee b) \wedge \\
(n1 \vee \overline{n3} \vee \overline{a}) \wedge (\overline{n1} \vee n3) \wedge (\overline{n1} \vee a) \wedge \\
(n3 \vee \overline{n5^*} \vee \overline{c}) \wedge (\overline{n3} \vee n5^*) \wedge (\overline{n3} \vee c) \wedge \\
(n4 \vee \overline{n11} \vee \overline{n3^*}) \wedge (\overline{n4} \vee n11) \wedge (\overline{n4} \vee n3^*) \wedge \\
(n11 \vee \overline{x} \vee \overline{y}) \wedge (\overline{n11} \vee x) \wedge (\overline{n11} \vee y) \wedge
\end{array}\right\} n6\text{'s side-input} \\
&\left.\begin{array}{l}
(n10 \vee \overline{n12} \vee n11) \wedge (\overline{n10} \vee n12) \wedge (\overline{n10} \vee \overline{n11}) \wedge \\
(n12 \vee \overline{y} \vee \overline{z}) \wedge (\overline{n12} \vee y) \wedge (\overline{n12} \vee z) \wedge
\end{array}\right\} n7\text{'s side-input} \\
&\left.\begin{array}{l} r \wedge \end{array}\right\} n8\text{'s side-input} \\
&\left.\begin{array}{l} s \end{array}\right\} n9\text{'s side-input}
\end{aligned}
$$

(c)

Fig. 5. An example for demonstrating CNF construction. (a) A cyclic circuit. (b) The validation model of loop $n6 - n7 - n8 - n9$ with its side-inputs. (c) The corresponding CNF of (b).

Fig. 7 shows the concept of constructing SAT-model for checking the combinationality of a cyclic circuit. $C_{1\_1} \sim C_{1\_n}$ are the CNF formulae of side-inputs in a loop constructed by *Loop Backtracking*. If all side-inputs of at least one loop can be non-controlling values simultaneously, the loop is a true loop and the SCC is non-combinational. Thus, all loops in an SCC are ORed together. Similarly, if any SCC is non-combinational, the cyclic circuit is also non-combinational. Therefore, all SCCs are ORed together.

D. Proposed Flow

This subsection summarizes the proposed two-stage algorithm in Fig. 8 for checking the combinationality of a cyclic circuit. Given a cyclic circuit C, we first identify loops using an efficient algorithm proposed in [11]. For every loop L, we search all the side-inputs of L for self-conflict detection. If a pair of SCSIs are found, the loop L will be filtered out. If all loops are self-conflict, the circuit is combinational. For loops that are not self-conflict, we perform *Loop Backtracking*. Finally, the CNF is constructed as suggested in Fig. 7 for SAT-solving. If SAT-solver returns UNSAT, the circuit C is combinational. Otherwise, the circuit is non-combinational.

Loop_Backtracking (SCC S)
For each non-self-conflict loop L in S
  For each side-input i of L
    If i has not been backtracked,
      $CNF \leftarrow CNF \wedge$ Backtracking_Side_Input (i).
Return CNF

Backtracking_Side_Input (Node n)
1. For each fanin f of n
    If f is in any loop and has been reached,
      use a virtual PI f^* to substitute f.
2. $CNF^* \leftarrow$ Construct CNF for n and its fanins.
3. For each fanin f of n that has not been backtracked
    Backtracking_Side_Input (f).
Return CNF^*

Fig. 6. The procedure of *Loop Backtracking*.

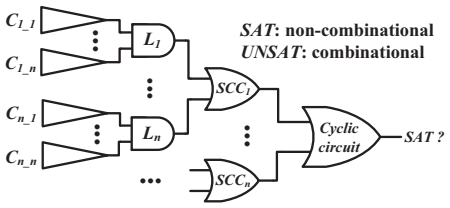


Fig. 7. The concept of constructing SAT-model in the proposed algorithm.

IV. Experimental Results

The proposed algorithm was implemented in C language within the ABC [4] environment on a 3.0 GHz Linux platform (CentOS 4.6). The benchmarks are from the IWLS 2005 [10] suite in AIG format, and they are first cyclified by the method in [5].

In the experiments, we also re-implemented the dual-rail model [1] for comparison. The experimental results are shown in TABLE I. Column 1 lists the cyclifiable benchmarks and their node counts, following by the columns showing the numbers of SCCs and loops in these benchmarks. $|Conf.|$ represents the number of loops that are detected as self-conflict. Columns 6 and 7 are the CPU time of [1] and ours measured in seconds, respectively. The last column shows the speedup of our approach. For example, $b17$ has 52920 nodes, 10 SCCs, and 20 loops. Three loops are detected as self-conflict by our approach. [1] spent 376.26 seconds to verify its combinationality while ours cost 0.11 seconds. The speedup is about 3420 times. For $s38417$, fortunately, all loops are detected as self-conflict, therefore, the speedup is significant.

According to TABLE I, our approach efficiently verify the combinationality of these circuits, even for the circuits with more than ten thousands of nodes. Our average speedup is about 4305 times.

V. Conclusions

Cyclic combinational circuits have some interesting features compared with acyclic ones. Checking combinationality of cyclic circuits is an important task in the synthesis of cyclic circuits. This paper proposes a two-stage algorithm that uses structural relations among side-inputs of a loop for checking combinationality of cyclic circuits. Compared with the state-of-the-art on performance for a set of benchmark, the proposed algorithm has a speedup of more than three orders of magnitude.

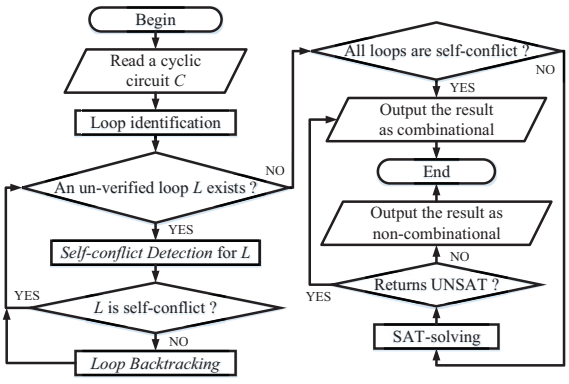


Fig. 8. The overall flow of the proposed algorithm.

TABLE I. The comparison of experimental results between [1] and our approach.

Benchmark	Node	SCC	Loop	Conf.	[1](s)	Ours(s)	Speedup
C432	209	1	9	0	0.15	0.01	15.00
C1908	414	1	160	0	0.05	0.03	1.67
rot	1063	2	3	1	0.04	0.01	4.00
i2c	1306	1	2	0	0.04	0.01	4.00
dalu	1740	3	3	0	0.90	0.01	90.00
s9234	1958	2	2	0	0.41	0.01	41.00
i10	2673	2	3120	2998	1.18	0.16	7.38
i8	3310	63	63	0	6.06	0.02	303.00
s38417	9219	8	8	8	78.86	0.01	7886.00
tv80	9609	5	213	1	34.92	0.16	218.25
systemcaes	13054	1	1	0	0.09	0.01	9.00
mem_ctrl	15641	1	21	0	20.21	0.09	224.56
usb_funct	15894	2	2	0	52.36	0.01	5236.00
aes_core	21513	1	1	0	47.01	0.01	4701.00
pci_bridge32	24369	3	5	0	212.11	0.02	10605.50
wb_conmax	48429	30	59	0	3758.86	0.14	26849.00
b17	52920	10	20	3	376.26	0.11	3420.55
Total					4589.51	0.82	
Ratio					5596.96	1	
Average							4305.99

References

[1] J. Backes, B. Fett, and M. D. Riedel, "The analysis of cyclic circuits with Boolean satisfiability," in *Proc. International Conference on Computer-Aided Design*, 2008, pp. 143–148.

[2] J. Backes and M. D. Riedel, "The synthesis of cyclic dependencies with Craig interpolation," in *Proc. International Workshop on Logic and Synthesis*, 2009, pp. 24–30.

[3] J. Backes and M. D. Riedel, "The synthesis of cyclic dependencies with Boolean satisfiability," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 17, pp. 44:1–44:24, 2012.

[4] Berkeley Logic Synthesis and Verificaiton Group. ABC: A System for Sequential Synthesis and Verification. [Online]. Available: http://www.eecs.berkeley.edu/∼alanmi/abc

[5] J.-H. Chen, "Making combinational circuit cyclic by identifying opportunities for introducing loops," *Master thesis, National Tsing Hua University, Taiwan*, 2013.

[6] Y.-C. Chen and C.-Y. Wang, "Fast detection of node mergers using logic implications," in *Proc. International Conference on Computer-Aided Design*, 2009, pp. 785–788.

[7] Y.-C. Chen and C.-Y. Wang, "Fast node merging with don't cares using logic implications," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 11, pp. 1827–1832, 2010.

[8] E. Craig, "Mathematical techniques in electronics and engineering analysis," *in Proc. of the IEEE*, vol. 52, no. 11, pp. 1390–1390, 1964.

[9] S. Edwards, "Making cyclic circuits acyclic," in *Proc. Design Automation Conference*, 2003, pp. 159–162.

[10] IWLS 2005 Benchmarks. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[11] D. B. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM J. Comput.*, vol. 4, no. 1, pp. 77–84, 1975.

[12] W. H. Kautz, "The necessity of closed circuit loops in minimal combinational circuits," *IEEE Trans. Computers*, vol. C-19, no. 2, pp. 162–164, 1970.

[13] O. Neiroukh, S. Edwards, and X. Song, "Transforming cyclic circuits into acyclic equivalents," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1775–1787, 2008.

[14] M. D. Riedel and J. Bruck, "The synthesis of cyclic combinational circuits," in *Proc. Design Automation Conference*, 2003, pp. 163–168.

[15] M. D. Riedel and J. Bruck, "Cyclic combinational circuits: Analysis for synthesis," in *Proc. International Workshop on Logic and Synthesis*, 2003, pp. 105–112.

[16] M. D. Riedel and J. Bruck, "Timing analysis of cyclic combinational circuits," in *Proc. International Workshop on Logic and Synthesis*, 2004, pp. 69–77.

[17] R. L. Rivest, "The necessity of feedback in minimal monotone combinational circuits," *IEEE Trans. Computers*, vol. C-26, no. 6, pp. 606–607, 1977.

[18] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of Reasoning*. Springer, 1983, pp. 466–483.