Synthesis and Verification of Cyclic Combinational Circuits

Jui-Hung Chen, Yung-Chih Chen†, Wan-Chen Weng, 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*—Prior works have demonstrated opportunities for achieving more minimized combinational circuits by introducing combinational loops during the synthesis. However, they achieved this by using a branch-and-bound technique to explore possible cyclic dependencies of circuits, which may not scale well for complex designs. Instead of using exploration, this paper proposes a formal algorithm using logic implication to identify cyclifiable structure candidates directly, or to create them aggressively in circuits. Additionally, we also propose a SAT-based algorithm to validate whether the formed loops are combinational or not. The effectiveness and scalability of the identification and validation algorithms are demonstrated in the experimental results performed on a set of IWLS 2005 benchmarks. As compared to the state-of-the-art algorithm, our validation algorithm produces speedups ranging from 2 to 2350 times.**

I. Introduction

Cyclic combinational circuits are a form of combinational circuit that, even in the presence of loops, behaves combinationally. For a cyclic circuit to behave combinationally, the value of every node in the circuit needs to be determined uniquely by its primary inputs (PIs) rather than any of their previous states. In other words, loops inside cyclic combinational circuits are not true loops from the perspective of a circuit's functionality. An acyclic circuit is certainly combinational since, while applying a certain input pattern to its PIs, the circuit computes fixed output values after being stable. Early works [17] [26] have shown that, to reach a minimized combinational circuit, we should not limit circuits to acyclic topologies. For example, in Fig. 1(a), implementing these six different output functions requires 7 gates. With cyclic structures, however, these functions can be realized with 6 gates as shown in Fig. 1(b).

Apart from the circuit area advantage, cyclic circuits offer other benefits. For example, [25] has shown that cyclic circuits can successfully remove logic gates off the critical path and benefit the circuit delay. All these advantages originate from the resource-sharing nature [29] of cyclic circuits.

Some prior works have been aware that combinational circuits might be cyclic, e.g., [19] [28] in circuit analysis, [14] [25] in timing analysis, and [22] in automatic test pattern generation (ATPG). However, most of these works intended to transform a cyclic circuit into its acyclic equivalent as in [11] [21] so that the traditional analysis for acyclic circuits could be applied effectively.

Recently, to pursue the advantages of cyclic combinational circuits, Riedel *et al.* [23] [24] have worked on the synthesis of cyclic combinational circuits in a general manner. They introduced loops to a circuit using a branch-and-bound algorithm and examined whether the circuit was combinational or not using a binary decision diagram (BDD)-based approach. However, the synthesis was operated on the Boolean expression in the functional level and did not guarantee that these functions could be mapped to a valid implementation [16]. Also, due to the inefficient nature of the branch-and-bound algorithm and potential memory explosion problem with the BDD-based approach, Backes *et al.* [1] [2] [4] switched to exploit the Craig interpolation [10] to synthesize cyclic circuits and used a satisfiability (SAT)-based approach to check the combinationality of circuits.

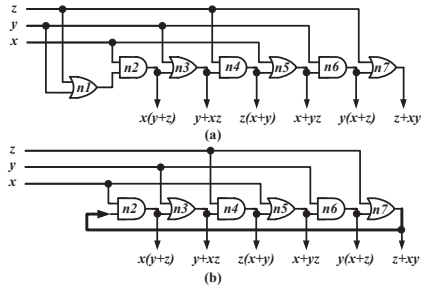

Fig. 1. An example for reaching more optimized circuits through introducing a combinational loop. (a) Acyclic implementation. (b) Cyclic version.

In the interpolation-based algorithm [2] [4], the authors explored cyclic dependencies by trying different support sets for each function in a branch-and-bound manner as well. Although this algorithm may have the ability to obtain the optimal solution from the support set size perspective, the algorithm could iterate until all elements in the support set became PIs, which requires great computational effort. Furthermore, since the Craig interpolation was applied, the size of the implementation strongly relies on the algorithm's capability of minimizing the interpolant [3].

In this paper, we propose a scalable algorithm that introduces combinational loops by merging two nodes. The merging and validation procedures are operated on the circuit level so that valid hardware implementation is guaranteed. We further transform this problem into identifying candidate *cyclic substitute nodes* (CSNs) or creating candidate *added cyclic substitute nodes* (ACSNs) in circuits. Since identifying these two kinds of nodes only requires three logic implications in the proposed algorithm, it increases the algorithm's efficiency. Additionally, we also propose a SAT-based algorithm to validate whether the formed loops are combinational or not.

Our experiments were conducted on IWLS 2005 benchmarks [15]. For the identification, the results show that our algorithm has the capability of both identifying existing cyclifiable structures and creating opportunities to turn an un-cyclifiable circuit into a cyclifiable one. For the validation, our algorithm results in speedups ranging from 2 to 2350 times compared to the previous approach [1].

The major contributions of this work are two-fold: First, this is the first cyclic combinational circuit synthesis algorithm that uses formal logic implication techniques on the circuit level considering observability don't care. Second, the proposed validation algorithm increases the efficiency of the validation by reducing the problem size for SAT solving. Combining the two contributions together makes the proposed algorithm scalable and capable of handling complex designs with up to tens of thousands of gates. Furthermore, with this technique, other research directions on the synthesis and analysis of cyclic combinational circuits are possible.

This work is supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 104-2220-E-155-001, 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.

978-1-4673-9094-1/15/$31.00 ©2015 IEEE

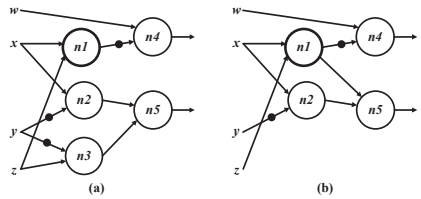

Fig. 2. (a) An original circuit. (b) The resultant circuit after merging $n1$ and $n3$.

II. Preliminaries

A. Background

Mandatory assignments (MAs) are necessary values assigned to specific gates or PIs in order to generate a test vector detecting the fault on a gate g. Let T denote the set of all the test vectors that can detect a stuck-at fault on g. A gate h is said to have an MA v (v is 0 or 1) if h has the same value v for all the test vectors in T. If one of these assignments contradicts another, no test vector exists for this fault. In consequence, the fault is untestable, and g is redundant [27]. The MAs can be obtained by the assignments for activating and propagating the fault effect. More MAs can be inferred by logic implications or recursive learning [18].

B. Node Merging

Definition 1: The effect of replacing a node n_t with another node n_s is to use n_s to drive the nodes originally driven by n_t. This effect is refer to as a misplaced-wire error.

If this error is undetectable, merging the two nodes will not affect the overall functionality of the circuit. Condition 1 is a sufficient condition for finding node mergers based on the mentioned idea.

Condition 1: [6] [7] *Let f denote an error of replacing n_t with n_s. If $n_s = 1$ and $n_s = 0$ are MAs for the stuck-at 0 and stuck-at 1 fault tests on n_t, respectively, f is undetectable.*

In Condition 1, n_t denotes a target node, and n_s denotes a substitute node. Merging n_t and n_s is equivalent to replacing n_t with n_s. Take Fig. 2 as an example, the functionalities of $n1$ and $n3$ only differ when $z = 1$ and $x = y$. Replacing $n3$ with $n1$ causes an error, which is the functional difference under $z = 1$ and $x = y$. However, $x = y$ implies $n2 = 0$, and $n2 = 0$ blocks the error effect so that the error effect is not observable at $n5$. Thus, the process of identifying the node mergers is transformed into performing two logic implications by Condition 1: deriving the MAs for the stuck-at 0 and stuck-at 1 fault tests on n_t.

We use Fig. 2 to illustrate the merger identification algorithm. For simplicity, the exemplary circuits in the rest of this paper are represented in and-inverter graphs (AIGs) [13]. Vertices in an AIG represent two-input AND gates; edges represent the connections among the gates; the dots on edges represent inverters. In Fig. 2(a), we compute MAs for the stuck-at 0 and stuck-at 1 fault tests on $n_t = n3$. The resultant MAs for the stuck-at 0 fault test on $n3$ are $\{n3 = 1,\ y = 0,\ z = 1,\ n2 = 1,\ x = 1,\ \mathbf{n1 = 1},\ n4 = 0,\ \mathbf{n5 = 1}\}$ and that for the stuck-at 1 fault test on $n3$ are $\{n3 = 0,\ n2 = 1,\ x = 1,\ y = 0,\ z = 0,\ \mathbf{n1 = 0},\ \mathbf{n5 = 0}\}$. These values can be computed by setting $n3 = 1$ (0) to activate the stuck-at 0 (1) fault effect, setting $n2 = 1$ to propagate the fault effects, and performing logic implications to derive additional MAs. According to Condition 1, $n1$ and $n5$ satisfy the requirements and can be mergers. However, only $n1$ was used as n_s to replace $n3$ in the works [6]–[9]. This is because using $n5$, a *cyclic* structure will be introduced. Fig. 2(b) shows the resultant circuit.

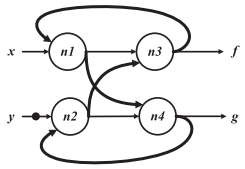

Fig. 3. An example illustrating the inputs of a loop.

III. Cyclifiable Structure Identification

In the last example, if we would like to employ $n5$ for forming a loop, more conditions are needed for confirming the circuit's combinationality. We refer to a candidate node that can be used to replace a target node to form combinational loops as a candidate *cyclic substitute node* (CSN). Thus, the problem of identifying cyclifiable structures in circuits becomes finding candidate CSNs in the given circuits and then validating their combinationality.

A. Candidate Cyclic Substitute Node Identification

Before introducing the proposed theorem, let us observe the behavior of feedback loops in circuits.

Definition 2: Given a circuit $c = (V, E)$, where V and E are the sets of vertices and edges belonging to c, respectively. A loop is defined as a set of nodes $l = \{n_1, n_2, ..., n_i, ..., n_n\}$ forming the feedback path, where $n_i \in V$.

For ease of discussion, edges are not included in our loop definition. Since the loop is defined as a set, the nodes in the set are distinct.

Definition 3: The inputs of a loop $l = \{n_1, n_2, ..., n_i, ..., n_n\}$ are defined as a set of nodes $p = \{m_1, m_2, ..., m_j, ..., m_m\}$ where m_j is a fanin to one of the n_i, but $m_j \notin l$.

For example, in Fig. 3, x and $\overline{y}$ are the inputs of the loop $l_1 = \{n1, n4, n2, n3\}$ while $n1$ and $n2$ are not. This is because $n1$ and $n2$ themselves are nodes forming l_1.

Definition 4: The input domain of a loop l is the space spanned by the inputs of l, i.e., the set of vectors which the input signals of l can attain.

For example, in Fig. 3, the input domain of the loop $l_1 = \{n1, n4, n2, n3\}$ is $\{(x = 0, \overline{y} = 0), (0, 1), (1, 0), (1, 1)\}$.

Before forming a loop by merging two nodes, we have to figure out the inputs of the loop in advance and ensure that it is impossible for these inputs to have input-noncontrolling values simultaneously. Since a formed loop can be considered as a path before the two nodes are merged, the inputs of the loop are the side inputs to the path. As a result, we need to ensure that the value changes on a target node n_t are never propagated to a substitute node n_s due to input-controlling value blocking on the paths. If that is the case, the loop formed by replacing n_t with n_s will remain combinational. Based on these observations, we propose Theorem 1 to preserve the combinationality of the formed loops.

Theorem 1: Let n_t denote a target node and n_s denote a substitute node in the transitive fanout cone of n_t. Replacing n_t with n_s forms a set of loops L. If the value changes on n_t are never propagated to n_s, L is combinational.

To check whether the value changes on n_t are propagated to n_s, we can try to generate an input pattern that sensitizes the paths from n_t to n_s. If such an input pattern exists, the paths are sensitized, and the formed loops are non-combinational. Otherwise, Theorem 1 is satisfied because no pattern propagates the value change, and the formed loops are combinational.

According to Theorem 1, we know that to identify a CSN, we need to test if there exists a pattern that sensitizes the paths connecting a n_t and n_s pair. If we would like to find all CSNs, we need to

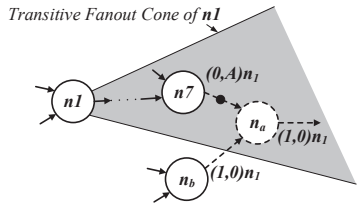

Fig. 4. An example showing the idea of a masking ACSN.

perform this test for every n_t and n_s pair, and this process would be very computation-intensive. Furthermore, we also have to keep the functionality intact when forming the loops.

For the purpose of examining both functionality and combinationality of the loops in a more efficient manner than the exhaustive search, we propose Condition 2 to efficiently identify candidate CSNs based on Condition 1 of the previous work [6] [7] and the proposed Theorem 1. These candidate CSNs have to be further validated for being CSNs because they fully satisfy Condition 1 but only partially satisfy Theorem 1.

Definition 5: To model the stuck-at fault effects in testing, two symbols, D and $\overline{D}$, are introduced. D ($\overline{D}$) means that the value is 1/0 (0/1), where 1 (0) is the fault-free value, and 0 (1) is the faulty value.

Condition 2: Let n_t denote a target node, and n_s denote a substitute node in the transitive fanout cone of n_t. Replacing n_t with n_s forms a set of loops L. If $n_s = 1$ and $n_t = D$ are MAs for the stuck-at 0 fault test on n_t, and $n_s = 0$ and $n_t = \overline{D}$ are MAs for the stuck-at 1 fault test on n_t, n_s is a candidate CSN.

To completely satisfy Theorem 1, we still need to ensure that the value changes on n_t cannot be propagated to n_s for other input assignments that make n_t's fault effects unobservable. We will discuss this issue in Section IV.

In summary, Condition 2 acts as an accelerating criterion for keeping a small amount of candidates from a large node pool. Through Condition 2, we can efficiently identify candidate CSNs using two logic implications: performing the stuck-at 0 and stuck-at 1 fault tests on n_t.

B. Candidate Added Cyclic Substitute Node Creation

In essence, the presence of CSNs in a circuit is strongly related to whether it is possible to share resources between different components in a circuit. Thus, sometimes the proposed candidate CSN identification procedure intrinsically fails to find candidate CSNs. However, for some circuits, it just needs adding a node, and then the added node can satisfy the condition for being a candidate CSN. We name the added nodes candidate *added cyclic substitute nodes* (ACSNs). In this subsection, we present two types of candidate ACSNs: *masking ACSNs* and *induced ACSNs*, which satisfy Condition 2 and could form a cyclifiable structure once any of them is added. The candidate ACSNs operate on the principle that finding a node n_s partially satisfying Condition 2, and then finding another node n_b blocking the value change from n_s so that the added node n_a, which is driven by n_s and n_b, can satisfy Condition 2.

For ease of discussion, we use the notation $n = (v_0, v_1)_m$ to represent that $n = v_0$ is an MA for the stuck-at 0 fault test on m, and that $n = v_1$ is an MA for the stuck-at 1 fault test on m. If v_0 (v_1) is expressed as "N", it means n is not an MA for the stuck-at 0 (stuck-at 1) fault test on m. If v_0 (v_1) is expressed as "A", it means v_0 (v_1) can be either any value of MA, i.e., 1, 0, D, and $\overline{D}$, or is not an MA, i.e., N. Using the notation, our objective now becomes finding an added node n_a driven by n_s and n_b such that $n_a = (1, 0)_{n_t}$ according to Condition 2.

1) Masking ACSNs: For example, for $n_t = n1$ in Fig. 4, assume we would like to introduce an added node n_a driven by $n_s = n7 = (0, A)_{n1}$, which has already partially satisfied Condition 2, and a node n_b, which can block the value change from propagation, such that $n_a = (1, 0)_{n1}$. This can be achieved by finding a node $n_b = (1, 0)_{n1}$ not in the transitive fanout cone of $n1$ and ANDing $\overline{n7}$. Since $n_a = \overline{n7} \cdot n_b = \overline{(0, A)_{n1}} \cdot (1, 0)_{n1} = (1, A)_{n1} \cdot (1, 0)_{n1} = (1, 0)_{n1}$ satisfies Condition 2, n_a is a candidate ACSN and could be used to replace $n1$ to form a combinational loop. Based on this observation, we propose Condition 3.

Condition 3: Let n_t denote a target node, and n_a denote an added node driven by a node n_s in the transitive fanout cone of n_t and a node n_b not in the transitive fanout cone of n_t. If $n_s = 1$ is an MA for the stuck-at 0 fault test on n_t and $n_b = (1, 0)_{n_t}$, n_a is a masking ACSN.

2) Induced ACSNs: Next, let us state the condition for creating induced ACSNs in Condition 4.

Condition 4: Let n_t denote a target node, and n_a denote an added node driven by a node n_s in the fanout cone of n_t and a node n_b not in the fanout cone of n_t. If $n_s = (1, N)_{n_t}$, $n_b = (1, N)_{n_t}$, and $n_s = 0$ is a value assignment implied from $n_b = 1$ for the stuck-at 1 fault test on n_t, then n_a is an induced ACSN.

Let us explain the validity of Condition 4. In Condition 4, ANDing $n_s = (1, N_s)_{n_t}$ and $n_b = (1, N_b)_{n_t}$ results in $n_a = (1, N_a)_{n_t}$, which has already satisfied half of Condition 2. Thus, if we can infer that $n_a = 0$ is also an MA for the stuck-at 1 fault test on n_t, i.e., $N_a = 0$, the other half of Condition 2 will also be satisfied. To determine whether $N_a = 0$, we can check whether all the test patterns T that can detect the stuck-at 1 fault on n_t make $n_a = 0$. If that is the case, $n_a = 0$ is an MA for the stuck-at 1 fault on n_t, which means $N_a = 0$. Based on the value of n_b, we classify T into $T_{n_b=0}$ and $T_{n_b=1}$, which are patterns generating $n_b = 0$ and $n_b = 1$, respectively. For $T_{n_b=0}$, it certainly generates $n_a = 0$ since 0 is a controlling value of n_a. On the other hand, if all the patterns in $T_{n_b=1}$ generate $n_s = 0$, then $n_a = 0$ will be an MA for the stuck-at 1 fault test on n_t. With this idea, Condition 4 is exploited to determine if all the test patterns in $T_{n_b=1}$ generate $n_s = 0$, i.e., $n_b = 1$ always implies $n_s = 0$.

In summary, to identify cyclifiable structures by either using a CSN or an ACSN to merge with the target node, only logic implication operations need to be performed, making the process efficient. This can be seen in the experimental results in Section VI.

IV. Cyclifiable Structure Validation

We next discuss how to validate the combinationality of a circuit from two aspects: inside a *strongly connected component* (SCC) — *intra-SCC validation*, and between one SCC and the others—*inter-SCC validation*.

A. Intra-SCC Validation

Based on Theorem 1, we propose an IntraSV model that can determine whether the fault effect on a target node n_t can be propagated to a substitute node n_s in Fig. 5. To determine if the fault effect on n_t propagates out, the only information we need is the nodes in the fanin cone of n_s. Therefore, the IntraSV model only consists of the nodes within the fanin cone of n_s, as Fig. 5(a) shows.

Next, we extend all wires and gates in this fanin cone to 2-bit wires and gates to accommodate two fault effects, D and $\overline{D}$, as shown in Fig. 5(b). The original logic 0 and 1 are encoded as 00 and 11, respectively; the fault effects D and $\overline{D}$ are encoded as 10 and 01, respectively. Since all signals now are 2-bit, the

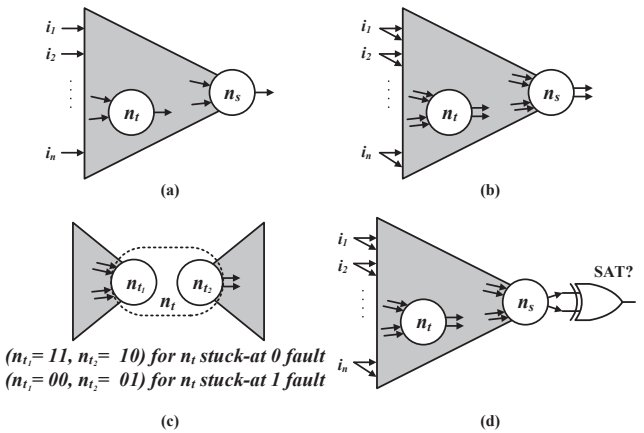

Fig. 5. The construction of the IntraSV model to validate the combinationality of an SCC. (a) Extracting the fanin cone of n_s. (b) Transforming signals from 1-bit to 2-bit. (c) Partitioning n_t into n_{t_1} and n_{t_2}. (d) Adding an XOR gate.

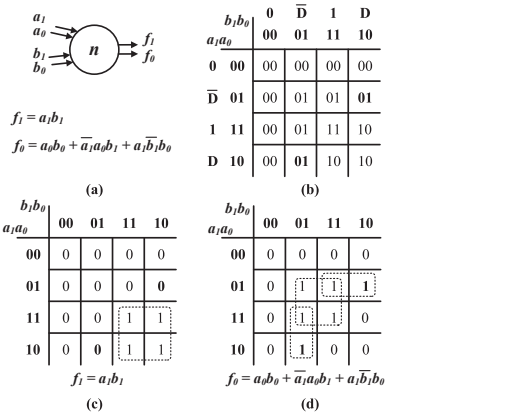

Fig. 6. The function derivation of a 4-input, 2-output AND node.

functionality of a 2-input AND node in the AIG has to be modified. As Fig. 6(a) illustrates, every 2-input AND node becomes a 4-input (a_1, a_0, b_1, b_0), 2-output (f_1, f_0) AND node, and the corresponding output functions f_1 and f_0 are listed below the new AND node n. In the Karnaugh map (K-map) of Fig. 6(b), the AND node performs normal AND operations in a pair-wise fashion among the 4 values 0, 1, D, and $\overline{D}$ except that $D \cdot \overline{D} = \overline{D}$. The $D \cdot \overline{D} = \overline{D}$ is intentionally set for detecting the self-masking fault effect. If there is any re-convergent fanout in between n_t and n_s, some fault effects may be cancelled and unobservable. This effect is illustrated in Fig. 7, where $n_t = n1$ and $n_s = n4$. Thus, if we allow $D \cdot \overline{D} = 0$ as usual, we will never distinguish whether this 0 comes from $n1$ or not. In fact, the 0 that is originated from $n1$ and propagated to $n4$ is not allowed. This is because if it happens, $n4$ will depend on previous states after merging $n1$ and $n4$, and the circuit will not be combinational. Through this slight modification to the K-map, however, we can still observe the fault effect on $n4$ even though $D \cdot \overline{D}$ occurs. Splitting the K-map in Fig. 6(b), the resultant K-maps for f_1 and f_0 are shown in Figs. 6(c) and 6(d), respectively. In this scheme, we use f_1 and f_0 to implement each 2-output AND node.

To activate the fault effect of the stuck-at 0 (1) fault, we need input patterns that generate $n_t = 1$ (0), encoded as 11 (00), but we also want to propagate the fault effect $n_t = D$ ($\overline{D}$), encoded as 10 (01), to n_s. Thus, we partition the node n_t into two copies, n_{t_1} and n_{t_2}, as shown in Fig. 5(c) and use two pairs of variables to represent their functions. With this scheme, we can restrict the PIs in the transitive fanin cone of n_{t_1} to patterns activating n_t to 1 (0)

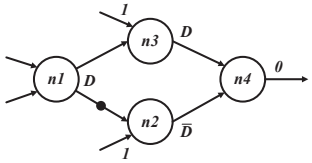

Fig. 7. An example for demonstrating the effect of f_1 and f_0 encoding.

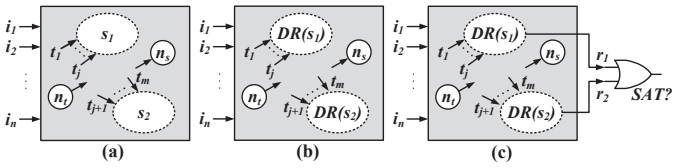

Fig. 8. The construction of the InterSV model. (a) Identifying SCCs s_1 and s_2. (b) Applying the dual-rail approach to s_1 and s_2. (c) ORing the outputs of $DR(s_1)$ and $DR(s_2)$.

by n_{t_1} and propagating the fault effect D ($\overline{D}$) of n_t by n_{t_2} within the same model for the intra-SCC validation. In the end, because we would like to detect a D or $\overline{D}$ on n_s, adding an XOR gate driven by the 2 outputs of n_s can achieve this objective, as Fig. 5(d) shows.

In summary, this IntraSV model is designed to find an assignment that activates n_t to 11 (00) and propagates the fault effect 10 (01) to n_s. This can be achieved by converting the IntraSV model to the CNF formula through Tseitin transformation [30] and applying the formula to a SAT-solver. If the SAT-solver returns SAT, it means that an assignment is found to simultaneously activate n_t and propagate the fault effect to n_s. Once such an assignment exists, the SCC formed by merging n_t and n_s violates Theorem 1 and is not combinational. Conversely, if no such assignment exists, the output of the XOR gate is 0, and the SAT-solver will return UNSAT meaning that the SCC is combinational.

B. Inter-SCC Validation

We propose a theorem to ensure that the currently formed SCC does not harm the combinationality of previously formed SCCs.

Definition 6: Given a circuit $c = (V, E)$, where V and E are the sets of vertices and edges belonging to c, respectively; and an SCC $s = (V_s, E_s)$, where V_s and E_s are the sets of vertices and edges belonging to s, respectively. The inputs to s are defined as a set of nodes $T_s = \{t_1, t_2, ..., t_i, ..., t_n\}$ where $t_i \in V$ but $t_i \notin V_s$.

Definition 7: The input domain D_s of an SCC s is defined as the space spanned by the inputs of s, i.e., the set of vectors which the input signals of s can attain.

Definition 8: An input domain D_s of an SCC s is said to be a valid input domain if all vectors in D_s do not cause s to have non-combinational behavior.

Theorem 2: Let n_t denote a target node and n_s denote a substitute node in the transitive fanout cone of n_t. Replacing n_t with n_s forms a new SCC s_{n+1}. Assume there are already n combinational SCCs $S = \{s_1, s_2, ..., s_i, ..., s_n\}$ in the circuit, and there are totally m input nodes $T = \{t_1, t_2, ..., t_j, ..., t_m\}$ to S. Let D_S and D_S^* denote the input domains of S before and after the replacement, respectively. If $D_S^* \subseteq D_S$, S remains combinational after s_{n+1} is formed.

According to Theorem 2, we understand that to ensure the newly formed SCC does not alter the circuit's combinationality, the input domain of all the previously formed SCCs should not be augmented after the merger. However, the effort for computing the input domain of all SCCs is equivalent to that for computing the output range of the sub-circuit excluding the SCCs, which is computation-intensive [12] [20]. Consequently, we propose an InterSV model that exploits

Identify_Cyclifiable_Structure (Circuit C)
For each node n_t in C in the DFS order from POs to PIs
1. Compute MAs for the stuck-at 0 fault test on n_t.
2. Compute MAs for the stuck-at 1 fault test on n_t.
3. $n_{csn} \leftarrow$ Find an n_s satisfying Condition 2.
4. If **Validate_Cyclifiable_Structure** (n_t, n_{csn}) returns true,
   (a) $C^* \leftarrow$ Merge n_t and n_{csn}.
   (b) Continue.
5. If n_t has a fanin that only fanouts to n_t,
   (a) $n_{acsn} \leftarrow$ Create a masking ACSN satisfying Condition 3.
   (b) If **Validate_Cyclifiable_Structure**(n_t, n_{acsn}) returns true,
      (i) $C^* \leftarrow$ Merge n_t and n_{acsn}.
      (ii) Continue.
   (c) $n_{acsn} \leftarrow$ Create an induced ACSN satisfying Condition 4.
   (d) If **Validate_Cyclifiable_Structure**(n_t, n_{acsn}) returns true,
      (i) $C^* \leftarrow$ Merge n_t and n_{acsn}.
      (ii) Continue.
Return C^*

Fig. 9. The algorithm for cyclifiable structure identification.

the dual-rail model proposed in [1] to check if the newly formed SCC influences other SCCs.

We use an example in Fig. 8 to demonstrate how we exploit the dual-rail model in the InterSV model. We apply the dual-rail model only within each SCC instead of within the entire circuit [1] to reduce computational effort. In Fig. 8(a), assume that two existing SCCs s_1 and s_2 have been identified, and $t_1 \sim t_j$ and $t_{j+1} \sim t_m$ are inputs of the two SCCs. Next, we create the dual-rail models $DR(s_1)$ and $DR(s_2)$ for s_1 and s_2, respectively, as shown in Fig. 8(b). Then, in Fig. 8(c), we OR the outputs of $DR(s_1)$ and $DR(s_2)$ together and assign different values to n_t and n_s to see if any of the dual-rail models produces 1. If that is the case, an invalid input pattern is discovered for at least one SCC when n_t and n_s have different values. This means that at least one SCC will be affected by the ODC if n_t and n_s are merged.

In summary, this InterSV model can be used to check if the newly formed SCC affects previous SCCs by finding an assignment that is not in the valid input domain of the SCCs. If such an assignment is found, one infers that this assignment is not in the original input domain, which violates Theorem 2.

V. Proposed Algorithm

We present the algorithms for cyclifiable structure identification and validation in Figs. 9 and 10.

We explain the identification algorithm in Fig. 9 first. Given a circuit C, we choose a target node n_t from C in depth-first search (DFS) order. Next, we compute MAs for the stuck-at 0 and stuck-at 1 fault tests on n_t. Among the obtained MAs, we figure out nodes $n_s = (1,0)_{n_t}$ or $n_s = (0,1)_{n_t}$ based on Condition 2. After the n_{csn} is obtained, we use the *Validate_Cyclifiable_Structure* procedure to validate the identified n_{csn}. The *Validate_Cyclifiable_Structure* procedure validates the given n_t and n_{csn} pair through the viewpoints of inside an SCC and between SCCs, as shown in Fig. 10.

Once the merger of n_t and n_{csn} fails, we further create ACSNs to form combinational loops for the same n_t. We first try to create a masking ACSN n_{acsn} because it does not need any additional logic implication based on Condition 3. If the resultant circuit is combinational, we choose the next n_t. Otherwise, we further try to create an induced ACSN using Condition 4. After performing the cyclifiable structure identification algorithm, we can obtain a circuit C^* containing combinational loops if any of CSNs or ACSNs are found.

VI. Experimental Results

We implemented the proposed algorithms within an ABC [5] environment using C language. Our experiments were conducted on a 3.0 GHz Linux platform (CentOS 4.6). The benchmarks are from

Validate_Cyclifiable_Structure (Node n_t, Node n_s)
1. $IsComb \leftarrow$ false.
2. If the intra-SCC validation in Section IV-A is passed,
   (a) If the inter-SCC validation in Section IV-B is passed,
      (i) $IsComb \leftarrow$ true.
Return $IsComb$

Fig. 10. The algorithm for cyclifiable structure validation.

TABLE I
The experimental results of CSN and ACSN identification.

Benchmark	N	N_t	$N_{csn+acsn}$	$\frac{N_{csn+acsn}}{N_t}$	$T(s)$
C432	209	27	60	2.22	0.02
C880	327	0	0	-	0.02
C499	400	0	0	-	0.03
C1908	414	6	23	3.83	0.07
C1355	504	0	0	-	0.06
C2670	717	0	0	-	0.10
C3540	1038	0	0	-	0.31
rot	1063	3	69	23.00	0.23
simple_spi	1079	0	0	-	0.14
i2c	1306	7	44	6.29	0.44
dalu	1740	4	5	1.25	3.47
C5315	1773	0	0	-	0.19
s9234	1958	10	24	2.40	0.53
C7552	2074	0	0	-	0.41
C6288	2337	0	0	-	0.49
i10	2673	7	19	2.71	3.53
systemcdes	3190	0	0	-	1.90
i8	3310	68	142	2.09	11.19
des_area	**4857**	**0**	**0**	**-**	**18.75**
s38417	**9219**	**8**	**16**	**2.00**	**1.43**
tv80	**9609**	**67**	**3640**	**54.33**	**66.20**
b20	12219	0	0	-	26.21
s38584	12400	0	0	-	25.29
b21	12782	0	0	-	33.47
systemcaes	13054	1	1	1.00	28.81
mem_ctrl	15641	1	9	9.00	322.99
usb_funct	15894	2	2	1.00	11.71
aes_core	21513	1	1	1.00	34.70
pci_bridge32	24369	6	16	2.67	29.41
wb_conmax	48429	30	160	5.33	72.52
b17	**52920**	**25**	**50**	**2.00**	**917.01**
des_perf	79288	0	0	-	42.53
Average				7.18	
Total					1653.80

the IWLS 2005 suite [15]. Every benchmark was transformed to the AIG format, and only its combinational portion was considered in the experiments. Additionally, a recursive learning technique [18] with the depth 1 was applied in our experiments to balance the computational efforts and completeness of the logic implications.

In the first experiment, we computed CSNs and ACSNs for each target node in the circuits using the proposed three conditions. The experimental results are summarized in Table I. Columns 1 and 2 list the benchmark and the node count in the AIG representation. Column 3 lists the number of target nodes that have a CSN or ACSN computed by our approach. Columns 4 and 5 show the total number of identified CSNs and created ACSNs, and the ratio to N_t. The last column lists the CPU time measured in seconds.

According to Table I, for the target nodes having cyclifiable structures, an average of 7.18 CSNs and ACSNs were obtained. We also realized that we cannot identify any N_t and $N_{csn+acsn}$ for some benchmarks, e.g., *des_area*. Additionally, we can identify CSNs and ACSNs in less than 28 minutes for all the benchmarks even though many benchmarks have node counts up to tens of thousands. The most time-consuming benchmark is *b17*, which costs about 16 minutes. Other benchmarks can be finished in less than 6 minutes. Consider two benchmarks with a similar size, e.g., *tv80* and *s38417*, *tv80* costs

TABLE II
THE COMPARISON ON VALIDATION ALGORITHMS.

Benchmark	Dual-rail [1]		Ours		Validation
	$T_v(s)$	$T_{total}(s)$	$T_v(s)$	$T_{total}(s)$	Speedup
C432	0.15	0.20	0.04	0.08	3.75
C1908	0.05	0.17	0.02	0.18	2.50
rot	0.04	0.34	0.02	0.25	2.00
i2c	0.04	0.57	0.01	0.42	4.00
dalu	0.19	4.16	0.04	3.39	4.75
s9234	0.41	1.07	0.01	0.53	41.00
i10	1.18	11.29	0.14	10.00	8.43
i8	6.06	18.84	1.99	12.64	3.05
s38417	**78.86**	**82.96**	**0.32**	**1.88**	**246.44**
tv80	34.92	404.82	1.39	182.37	25.12
systemcaes	0.09	35.13	0.02	27.81	4.50
mem_ctrl	20.21	467.62	0.70	389.86	28.87
usb_funct	52.36	66.74	0.04	11.69	1309.00
aes_core	47.01	88.43	0.02	33.62	2350.50
pci_bridge32	212.11	260.14	0.14	37.96	1515.07
wb_conmax	**3758.86**	**3848.85**	**11.60**	**82.59**	**324.04**
b17	376.26	1332.06	3.01	910.95	125.00
Average					352.82
Total	4588.80	6623.40	19.50	1706.22	

much more CPU time. This is because *tv80* contains more cyclifiable structures than *s38417* in terms of the number of N_t and $N_{csn+acsn}$. In summary, from the efficiency viewpoint, the CPU time for the cyclifiable structure identification is acceptable, and the proposed approach is scalable for complex circuits.

In the second experiment, we compared the efficiency of the proposed cyclifiable structure validation algorithm mentioned in Section IV with that of the dual-rail validation approach in [1]. We re-implemented their validation approach so that it could be integrated with our identification algorithm. We only select the benchmarks containing cyclifiable structures from Table I. This experiment follows the algorithm presented in Fig. 9 for the column "Ours". The procedure "Validate_Cyclifiable_Structure" is replaced with the re-implemented version of [1] for the column "Dual-rail". Only the CPU time used in the "Validate_Cyclifiable_Structure" procedure is calculated. In Table II, Columns 2 and 3 list the time for the dual-rail validation approach and the total elapsed time for the whole identification and validation process. Columns 4 and 5 list the corresponding results of the proposed algorithm, respectively. The last column lists the speedup in terms of the time spent on validation. For example, *s38417* took 78.86 seconds using the dual-rail validation approach, and the time for the total process is 82.96 seconds. Using the proposed validation algorithm, however, it only needed 0.32 seconds to finish the validation, and the total time is 1.88 seconds. Hence, the speedup is 246.44 times.

According to Table II, the proposed algorithm achieved speedups ranging from 2 to 2350 times in validation time. This speedup helps reduce the large amount of time spent on complicated circuits. Although the speedup is minimal for the smaller circuits, it brings a tremendous advantage in efficiency for larger circuits. Take *wb_conmax* as an example, the required 3758.86 seconds in validation are reduced to 11.60 seconds, or 324.04 times speedup. Accomplished with the proposed identification algorithm, the whole procedure for forming combinational loops took no more than 29 minutes for all the benchmarks, making it feasibly scalable.

VII. CONCLUSION

In this work, we propose an algorithm that introduces combinational feedback loops by merging two nodes. The algorithm can identify cyclifiable structure candidates in circuits using efficient logic implications. Additionally, we introduce new techniques to validate the cyclifiable structures before the loops are formed. On average, the new techniques bring more than two orders of magnitude in speedup for validation process. The experimental results show the scalability of the identification algorithm and the efficiency of the validation algorithm for introducing combinational loops in circuits.

REFERENCES

[1] J. Backes *et al.*, "The analysis of cyclic circuits with Boolean satisfiability," in *Proc. ICCAD*, 2008, pp. 143–148.

[2] J. Backes *et al.*, "The synthesis of cyclic dependencies with Craig interpolation," in *Proc. IWLS*, 2009, pp. 24–30.

[3] J. Backes *et al.*, "Reduction of interpolants for logic synthesis," in *Proc. ICCAD*, 2010, pp. 602–609.

[4] J. Backes *et al.*, "The synthesis of cyclic dependencies with Boolean satisfiability," *ACM TDAES*, vol. 17, pp. 44:1–44:24, 2012.

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

[6] Y.-C. Chen *et al.*, "Fast detection of node mergers using logic implications," in *Proc. ICCAD*, 2009, pp. 785–788.

[7] Y.-C. Chen *et al.*, "Fast node merging with don't cares using logic implications," *IEEE TCAD*, vol. 29, no. 11, pp. 1827–1832, 2010.

[8] Y.-C. Chen *et al.*, "Node addition and removal in the presence of dont cares," in *Proc. DAC*, 2010, pp. 505–510.

[9] Y.-C. Chen *et al.*, "Logic restructuring using node addition and removal," *IEEE TCAD*, vol. 31, no. 2, pp. 260–270, 2012.

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

[11] S. Edwards, "Making cyclic circuits acyclic," in *Proc. DAC*, 2003, pp. 159–162.

[12] A. Gupta *et al.*, "Sat-based image computation with application in reachability analysis," in *Proc. FMCAD*, 2000, pp. 391–408.

[13] L. Hellerman, "A catalog of three-variable or-invert and and-invert logical circuits," *IEEE Trans. Electronic Computers*, vol. EC-12, no. 3, pp. 198–223, 1963.

[14] Y.-C. Hsu *et al.*, "Finding the longest simple path in cyclic combinational circuits," in *Proc. ICCAD*, 1998, pp. 530–535.

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

[16] J.-H. Jiang *et al.*, "On breakable cyclic definitions," in *Proc. ICCAD*, 2004, pp. 411–418.

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

[18] W. Kunz *et al.*, "Recursive learning: a new implication technique for efficient solutions to CAD problems-test, verification, and optimization," *IEEE TCAD*, vol. 13, no. 9, pp. 1143–1158, 1994.

[19] S. Malik, "Analysis of cyclic combinational circuits," *IEEE TCAD*, vol. 13, no. 7, pp. 950–956, 1994.

[20] J. Moondanos *et al.*, "Clever: Divide and conquer combinational logic equivalence verification with false negative elimination," in *Proc. CAV*, 2001, pp. 131–143.

[21] O. Neiroukh *et al.*, "Transforming cyclic circuits into acyclic equivalents," *IEEE TCAD*, vol. 27, no. 10, pp. 1775–1787, 2008.

[22] A. Raghunathan *et al.*, "Test generation for cyclic combinational circuits," in *Proc. International Conference on VLSI Design*, 1995, pp. 104–109.

[23] M. D. Riedel *et al.*, "The synthesis of cyclic combinational circuits," in *Proc. DAC*, 2003, pp. 163–168.

[24] M. D. Riedel *et al.*, "Cyclic combinational circuits: Analysis for synthesis," in *Proc. IWLS*, 2003, pp. 105–112.

[25] M. D. Riedel *et al.*, "Timing analysis of cyclic combinational circuits," in *Proc. IWLS*, 2004, pp. 69–77.

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

[27] M. Schulz *et al.*, "Advanced automatic test pattern generation and redundancy identification techniques," in *Proc. International Fault-Tolerant Computing Symposium*, 1988, pp. 30–35.

[28] T. Shiple *et al.*, "Constructive analysis of cyclic circuits," in *Proc. European Design and Test Conference*, 1996, pp. 328–333.

[29] L. Stok, "False loops through resource sharing," in *Proc. ICCAD*, 1992, pp. 345–348.

[30] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Stud. Constr. Math. and Math. Logic*, 1968, pp. 115–125.