Dependent-Latch Identification in Reachable State Space

Chen-Hsuan Lin, Chun-Yao Wang, *Member, IEEE*, and Yung-Chih Chen

***Abstract*—The large number of latches in current digital designs increases the complexity of formal verification and logic synthesis, since an increase in latch numbers leads to an exponential expansion of the state space. One solution to this problem is to find the functional dependences among these latches. With the information of functional dependences, these latches can be identified as dependent or essential latches, and the state space can be constructed using only the essential latches. Although much research has been devoted to exploring the functional dependences among latches using binary-decision-diagram (BDD)-based symbolic algorithms, this issue is still unresolved for large sequential circuits. In this paper, we propose a heuristic to identify the dependent latches based on the state-of-the-art work. In addition, our proposed approach detects sequential functional dependences existing in the reachable state space only. The sequential functional dependences can identify additional dependent latches after a specific time frame in order to achieve additional reduction of the state space. Experimental results show that this approach can deal with large sequential circuits with up to 9000 latches in a reasonable time while simultaneously identifying their combinational and sequential dependent latches. For instance, with s13207 in ISCAS'89, 23% of the latches are identified as combinational dependent latches, and an additional 13% of the latches are identified as sequential dependent latches. For the reachability analysis of s13207, with the benefits of dependent-latch identification, 70.70% of the BDD size and 73.32% of the CPU time can be reduced within the same time frame. Furthermore, 2890.76% more states can be reached under the 600 000-s run-time limit.**

***Index Terms*—Dependent-latch identification, reachability analysis.**

I. Introduction

IN THE FORMAL verification of sequential systems, a main task—called sequential equivalence checking (SEC)—is to test the equivalence of two given sequential circuits. Examining two sequential circuits, S_1 and S_2, for equivalence can be reduced to a reachability analysis by building a product machine of the two sequential circuits (called a miter in [4]). To build a product machine, two sequential circuits S_1 and S_2 are combined with XOR gates at each pair of primary outputs. Therefore, if these two sequential circuits have many latches, the corresponding product machine will have more latches. Since the state space of a product machine is the Cartesian product of the state space of S_1 and S_2 and it grows exponentially with the increase of the number of latches in it, the corresponding reachability analysis of the machine will become more difficult. Furthermore, the performance of SEC strongly depends on the reachability analysis [5] of the product machine. In addition to SEC, many other tasks for sequential circuits, such as property checking and model checking [21], can also be reduced to a reachability analysis. Hence, reachability analysis plays an important role in the formal verification of sequential systems, and its efficiency dramatically influences the performance of formal verification for sequential circuits.

To improve the efficiency of reachability analysis, researchers have proposed many approaches, such as early quantification [13], approximative state-space traversal [8], [10], [22], and functional dependences [11], [27]. Early quantification is a method for quantifying the binary-decision-diagram (BDD) [2] variables of primary inputs (PIs) and pseudo-primary inputs (PPIs) earlier to alleviate the effort of transition relation construction, which is the bottleneck of reachability analysis. Approximative state-space traversal is proposed to perform under- or overapproximation on BDDs to reduce the BDD size and the complexity of BDD operations during reachability analysis. Functional dependences, which this paper focuses on, are used to identify dependent latches in a machine. Since some latches in a system might functionally depend on the other latches, it is possible that not all latches in a system have to be considered during reachability analysis. Therefore, the identification of dependent latches, which depend on the other latches within the system for their functionality, plays an important role in reducing the state space of a product machine. It also helps to minimize the number of latches in a sequential circuit.

To identify the dependent latches in a sequential circuit, the functional dependences between these latches must be identified. In this paper, the functional dependences are the relationships between these latches, and their corresponding Boolean functions are called *dependence functions*. With information about dependence functions obtained, dependent latches can be replaced by other latches. Hence, a sequential circuit can be optimized by removing these dependent latches without changing the circuit's functionality [18]. In addition, disregarding dependent latches can alleviate the problem of explosive BDD size expansion during BDD-based reachability analysis [11]. The dependence function also has a wide range of electronic-design-automation applications, such as register-transfer-level synthesis [18], [24], BDD minimization [11], and logic synthesis [16].

Manuscript received August 29, 2008; revised January 13, 2009. Current version published July 17, 2009. This work was supported in part by the National Science Council of R.O.C. under Grants NSC 97-2220-E-007-042 and NSC 97-2220-E-007-034. This paper was recommended by Associate Editor W. Kunz.

The authors are with the Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan (e-mail: Adonis@nthucad.cs.nthu.edu.tw; wcyao@cs.nthu.edu.tw; ycchen@cs.nthu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2020720

Equivalence relation ($F_i \equiv F_j$) and opposition relation ($F_i = \neg F_j$) are examples of some typical functional dependences. References [12] and [27] have proposed algorithms to identify them. However, there may exist many other relations among these latches. For instance, AND relation ($F_i = F_j \bullet F_k$) is another relation. An approximate approach in [14] directly extracts dependence functions from the latches' transition functions by using BDDs. However, the scalability of the approach is still restricted to the BDD size. As a result, this approach might not efficiently identify all functional dependences among latches. Therefore, another approach [17] detects the functional dependence by using a SAT solver with the Craig interpolation theorem [7]. With the recent great advances in SAT solvers [9], [19], this approach can identify more functional dependences among latches.

Although [17] exploited a SAT solver to find more functional dependences among latches, the identified dependence functions may not be precise enough because of using maximal input support candidates. Furthermore, it only reports which latches have dependence functions but does not explicitly indicate the dependent latches. Thus, the identified results cannot be used directly for further applications. On the other hand, since [17] extracts the functional dependence from the latches' transition functions, its functional dependence is a combinational functional dependence, which holds in the whole state space. As for the sequential functional dependence, which holds only in reachable state space, [17] cannot explore it. The sequential functional dependence is capable of identifying additional dependent latches in a circuit after a specific time frame. Since the initial state is unrestricted in this paper, the sequential functional dependences identified are held in the overapproximated reachable state space. For ease of reading, the term "overapproximated" is omitted in the succeeding discussion.

The contributions of this paper are as follows: 1) An ordered destroyed cost heuristic is derived to minimize the input support candidates such that as many dependent latches are identified as possible. The corresponding dependent functions among the latches are derived as well. 2) An efficient method for discovering sequential functional dependences among latches by exploiting the incremental SAT technique and the early detection of dependent latches is proposed. As a result, more dependent latches can be identified within each time frame.

The rest of this paper is organized as follows: Section II introduces the preliminaries. Section III uses simple examples to demonstrate the effect of the input support candidate selection and to describe the limitations of the state-of-the-art. Section IV introduces the proposed heuristic to find combinational functional dependences as well as sequential functional dependences with greater precision. The experimental results of dependent-latch identification and its application to reachability analysis are reported in Section V. Section VI concludes this paper.

II. Preliminaries

This section introduces the fundamental concepts of functional dependence and the basic components of this work [7], [14], [17].

A. Functional Dependence of Latches

First, the dependence function of a latch is introduced.

Definition 1: A latch's transition function is a Boolean function $f(X)$ whose input domain X consists of the PIs and PPIs. It determines the next state value of this latch. A PPI is the output signal of a latch, and a pseudo-primary output (PPO) is the input signal of a latch.

Definition 2: Given a latch's transition function $r(X) : B^m \rightarrow B$, $B = \{0, 1\}$, and a vector of other latches' transition functions $S(X) = \langle s_1(X), \ldots, s_n(X)\rangle$, where $s_i(X) : B^m \rightarrow B$ for $i = 1, \ldots, n$ over the same input domain $\{X \in B^m | X = \langle x_1, \ldots, x_m\rangle\}$, the total number of PIs and PPIs (latches) is m. r combinationally functionally depends on S if there is a Boolean function $d : B^n \rightarrow B$, called the dependence function, such that $r(X) = d(s_1(X), \ldots, s_n(X))$. The latch r is called a dependent latch, and the latches in vector S are called base latches.

Note that the dependent latch might depend only on a subset of the base latch set $\{s_1(X), \ldots, s_n(X)\}$ when the dependence function is written as $r(X) = d(s_1(X), \ldots, s_n(X))$.

B. Existence of Functional Dependence

A necessary and sufficient condition to examining the existence of functional dependence among latches is described as follows.

Theorem 1 [14]: Given the transition function of a dependent latch r and the transition functions of base latches S, let $d^0 = \{Y \in B^n | Y = S(X) \text{ and } r(X) = 0, X \in B^m\}$ and $d^1 = \{Y \in B^n | Y = S(X) \text{ and } r(X) = 1, X \in B^m\}$. The dependence function d exists if and only if $d^0 \cap d^1$ is empty. Therefore, d^0, d^1, and $B^n \setminus (d^0 \cup d^1)$ could be considered as the off, on, and don't-care sets of d, respectively. In brief, when $d^0 \cap d^1 = \phi$, $r(X) = d(S(X))$ is always true for all input combinations of X.

Theorem 1 can be used to examine whether the dependence function d exists or not.

C. Exploration of Functional Dependence by SAT Solvers

Before using SAT solvers, the circuit netlist needs to be transformed to the conjunctive normal form (CNF), which is the input format of SAT solvers [9], [19]. The following paragraphs will introduce the *dependence function network*, which is used to examine the existence of functional dependence in the circuit and explore the corresponding dependence function. The meaning of the network will then be explained. Finally, there will be an illustration of how to transform this network into a SAT problem.

1) Dependence Function Network: Lee *et al.* [17] define a general network, as seen in Fig. 1, to establish whether the dependence function exists by using a SAT solver. This method will extract the combinational part of a sequential circuit with l latches. This combinational part includes each latch's transition function with the PIs, PPIs, and PPOs as the signals (the primary outputs will be ignored in this paper). $x_i, i = 1, \ldots, m$, is one of PIs or PPIs, and $y_j, j = 1, \ldots, l$, is the function of

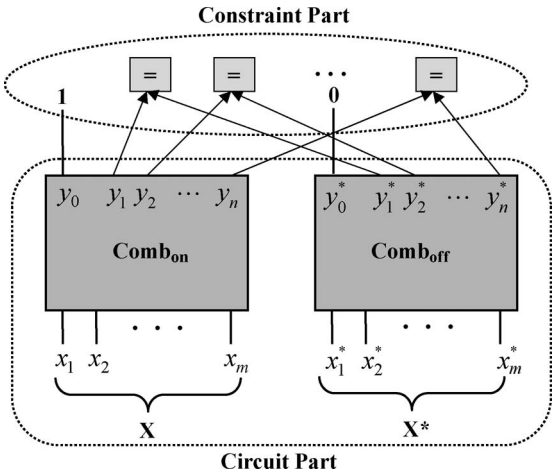

Fig. 1. Dependence function network for SAT solvers [17].

the PPO. Each y_j represents the next state transition function of latch j; therefore, this latch's next state value is $y_j(x_1, \ldots, x_m)$. The combinational part of the circuit is instantiated into two copies called $Comb_{\rm on}$ and $Comb_{\rm off}$ to form the circuit part of the network. For every variable v in $Comb_{\rm on}$, there is a counterpart v^* in $Comb_{\rm off}$. The constraint part of the network selects $(n+1)$ out of l latches that are considered in the circuit, $(n+1) \leq l$. With y_j and y_j^*, $j = 0, \ldots, n$, in the constraint part, $y_0 = 1$ is set as the constraint in $Comb_{\rm on}$, and $y_0^* = 0$ is set in $Comb_{\rm off}$. This action results in the domain of X (respectively X^*) being restricted to the subdomain leading $y_0 = 1$ (respectively $y_0^* = 0$). Similarly, the domain of $\langle y_1, \ldots, y_n \rangle$ (respectively $\langle y_1^*, \ldots, y_n^* \rangle$) is restricted to the subdomain based on the constrained domain of X (respectively X^*).

2) Dependence Function Network Operation: To check whether a dependence function d exists among a dependent latch r and the base latches $\langle s_1, \ldots, s_n \rangle$, y_0 and $\langle y_1, \ldots, y_n \rangle$ in this network can be regarded as the dependent latch r and the base latches $\langle s_1, \ldots, s_n \rangle$, respectively. Then, the constrained subdomain of $\langle y_1, \ldots, y_n \rangle$ can be taken as the on set d^1 and the constrained subdomain of $\langle y_1^*, \ldots, y_n^* \rangle$ as the off set d^0. Therefore, if the subdomain of $\langle y_1, \ldots, y_n \rangle$ and the subdomain of $\langle y_1^*, \ldots, y_n^* \rangle$ overlap, it indicates that $d^0 \cap d^1$ is not empty and the dependence function d does not exist, according to Theorem 1.

3) Dependence Function Network Transformation to a SAT Problem: The circuit parts $Comb_{\rm on}$ and $Comb_{\rm off}$ of the network in Fig. 1 can be converted to the CNFs $C_{\rm on}$ and $C_{\rm off}$, respectively [1]. Similarly, the constraint part can be converted to the following CNF: $y_0 \wedge \neg y_0^* \wedge (y_1 \equiv y_1^*) \wedge \cdots \wedge (y_n \equiv y_n^*)$. Therefore, the complete CNF of the dependence function network is

$$C_{\rm network} = C_{\rm on} \wedge C_{\rm off} \wedge y_0 \wedge \neg y_0^* \wedge (y_1 \equiv y_1^*) \wedge \cdots \wedge (y_n \equiv y_n^*)$$

where $(y_j \equiv y_j^*)$ stands for $(y_j \vee \neg y_j^*) \wedge (\neg y_j \vee y_j^*)$.

Finally, SAT solvers can be used to examine whether $C_{\rm network}$ is satisfiable or unsatisfiable.

1) **Satisfiable**: There exists one assignment of $\langle y_1, \ldots, y_n \rangle$ and $\langle y_1^*, \ldots, y_n^* \rangle$ such that $(y_0 = 1)$ and $(y_0^* = 0)$ are both true. That is, $d^0 \cap d^1$ is not empty, and the dependence function d does not exist.
2) **Unsatisfiable**: No assignment of $\langle y_1, \ldots, y_n \rangle$ and $\langle y_1^*, \ldots, y_n^* \rangle$ can be found; therefore, $d^0 \cap d^1$ is empty, and the dependence function d exists.

The aforementioned summaries are based on the following theorem proposed in [17].

Theorem 2 [17]: Given the transition function $r(X)$ of a dependent latch and the transition functions $\langle s_1(X), \ldots, s_n(X) \rangle$ of base latches, a dependence function d exists between the dependent latch and base latches if and only if the corresponding $C_{\rm network}$ is unsatisfiable.

D. Determination of Dependence Function by Craig Interpolation

Theorem 3 [7]: Craig Interpolation Theorem. In the two CNFs C_a and C_b with the common input variables $\langle v_0, \ldots, v_n \rangle$, if $C_a \wedge C_b$ is unsatisfiable, there exists a Boolean formula I only referring to the common input variables $\langle v_0, \ldots, v_n \rangle$ with the property that $C_a \Rightarrow I$ and $I \Rightarrow \neg C_b$.

This Boolean formula I is called the *interpolant* of C_a and C_b. The interpolant can be constructed in linear time from the refutation proof [15], [20], [23], and current SAT solvers [9], [19] can easily produce it from an unsatisfiable instance.

Theorem 4 [17]: If the CNF $C_{\rm network} = C_{\rm on} \wedge C_{\rm off} \wedge y_0 \wedge \neg y_0^* \wedge (y_1 \equiv y_1^*) \wedge \cdots \wedge (y_n \equiv y_n^*)$ is unsatisfiable, we partition $C_{\rm network}$ into C_a and C_b (i.e., $C_{\rm network} = C_a \wedge C_b$) with $C_a = C_{\rm on} \wedge y_0$ and $C_b = C_{\rm off} \wedge \neg y_0^* \wedge (y_1 \equiv y_1^*) \wedge \cdots \wedge (y_n \equiv y_n^*)$. The common input variables of C_a and C_b are $Y = \langle y_1, \ldots, y_n \rangle$. Because $C_{\rm network}$ is unsatisfiable, there exists an interpolant formula I only referring to the common variables Y, and $I(Y)$ can be taken as the dependence function d.

The detailed proof of [17, Th. 4] shows that $I(Y)$ must be an overapproximation of $d^1(Y)$ and must be disjoint from $d^0(Y)$. Therefore, $I(Y)$ is the valid dependence function such that $r = I(s_1, \ldots, s_n)$ is always true.

III. Limitations of the State-of-the-Art

This section discusses the limitations of the state-of-the-art [17] using several examples.

A. Difference Between Functional Dependences and Dependent Latches

Previous work [17] explored dependence functions among latches, but the study failed to adequately address dependent-latch identification. Therefore, additional processes are necessary to identify the dependent latches.

B. Effect of Input Support Candidate Selection

To identify all functional dependences in a sequential circuit, the previous work takes all the other latches as input support candidates—referred to as maximal input support candidates in this paper—to explore the dependence function of a latch. However, with maximal input support candidates, the dependence functions obtained in the previous work may contain some

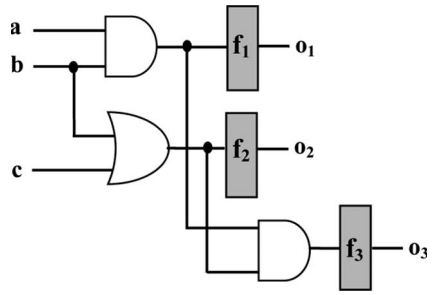

Fig. 2. Example demonstrating that maximal input support candidates lead dependence functions to contain redundant input supports.

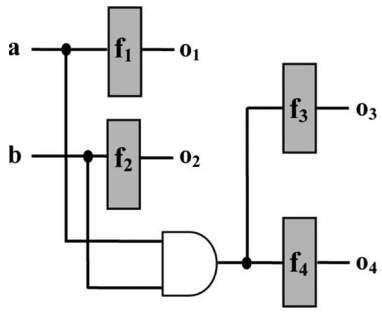

Fig. 3. Example demonstrating that maximal input support candidates lead dependence functions to rely on dependent latches.

redundant input supports and be reliant on dependent latches. These results do not benefit dependent-latch maximization. Two simple examples, shown in Figs. 2 and 3, are used to demonstrate the effects of input support candidate selection.

1) Dependence Functions Containing Redundant Input Supports: Fig. 2 shows a sequential circuit. In order to examine whether f_3's functional dependence exists, [17] takes $\{f_1, f_2\}$, the maximal input support candidates, as its input support candidates. As a result, it will get a dependence function $f_3 = f_1 \bullet f_2$, as can be observed from Fig. 2. However, if f_3's input support candidates are restricted to $\{f_1\}$, a more simplified but not intuited dependence function $f_3 = f_1$ is found. This is because $f_3 = f_1 \bullet f_2 = (ab) \bullet (b + c) = ab + abc = ab = f_1$.

2) Dependence Functions Relying on Dependent Latches: Fig. 3 shows a sequential circuit. In order to examine whether f_3's functional dependence exists using the method proposed in [17], $\{f_1, f_2, f_4\}$ are taken as input support candidates. The results give a dependence function $f_3 = f_4$. Dealing in the same way with f_1, f_2, and f_4, f_1 and f_2 are regarded as independent latches and will also give the result of $f_4 = f_3$. According to the dependence functions of f_3 and f_4 explored by [17], only one of them can be identified as a dependent latch; therefore, the other one should be an essential latch. In this paper, however, the input support candidates of f_3 and f_4 will be restricted to $\{f_1, f_2\}$, and their corresponding dependence functions $f_3 = f_1 \bullet f_2$ and $f_4 = f_1 \bullet f_2$ will be obtained. According to these dependence functions, f_3 and f_4 can be simultaneously identified as dependent latches.

C. Dependent-Latch Identification by the Results of [17]

Another example, shown in Fig. 4, can be used to expose the problems that may occur when using the results of [17] to identify dependent latches in the circuits. This example will also

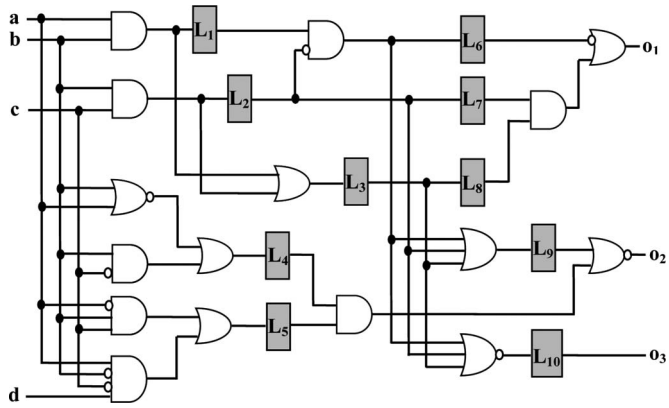

Fig. 4. Demonstrating example.

L_1's input support candidates = $\{L_2, \ldots, L_{10}\}$
L_2's input support candidates = $\{L_1, L_3, \ldots, L_{10}\}$
L_3's input support candidates = $\{L_1, L_2, L_4, \ldots, L_{10}\}$
L_9's input support candidates = $\{L_1, \ldots, L_8, L_{10}\}$
L_{10}'s input support candidates = $\{L_1, \ldots, L_9\}$

Fig. 5. Previous work [17] uses the maximal input support candidates to explore the dependence functions.

be used to describe the proposed heuristic in the next section. Fig. 4 shows a sequential circuit with ten latches $\{L_1, \ldots, L_{10}\}$ and four PIs $\{a, b, c, d\}$. To identify dependent latches, each latch's dependence function must be explored first, if it exists. In this example, only five latches $\{L_1, L_2, L_3, L_9, L_{10}\}$ have functional dependences.[1] These dependence functions' input support candidates are shown in Fig. 5. Although these five latches can be replaced by their base latches according to the dependence functions, they cannot be simultaneously regarded as dependent latches due to their circular dependences. Therefore, some of these five latches have to be selected as essential latches such that the circular dependences can be broken. This selection will influence the number of dependent latches identified. Nevertheless, since [17] always includes all other latches as the input support candidates, the dependence functions explored might have redundant input supports or rely on dependent latches. In this example, the exact input support set of L_1, L_2, and L_3 are $\{L_3, L_5\}$, $\{L_3, L_4\}$, and $\{L_1, L_2\}$, respectively. With the independent latches $\{L_4, L_5\}$, L_3 can be chosen as an essential latch, and then, $\{L_1, L_2\}$ can be identified as dependent latches. However, if L_1's dependence function includes a redundant input L_2 and L_2's dependence function includes a redundant input L_1, their input support sets become $L_1 = \{L_2, L_3, L_5\}$ and $L_2 = \{L_1, L_3, L_4\}$. In this situation, no matter how essential latches are chosen, only one dependent latch can be identified. In addition, L_9's input support candidate set includes L_{10}, and L_{10}'s input support candidate set includes L_9; thus, [17] explores their corresponding dependence functions, $L_9 = \neg L_{10}$ and $L_{10} = \neg L_9$. In this situation, only one of L_9 and L_{10} can be regarded as a dependent latch. As a result, [17] identifies fewer redundant latches. Therefore,

[1]Referring to Fig. 7, five of ten latches in this example are independent latches. The remaining five latches have functional dependences.

the next section will propose a heuristic to minimize the input support candidates in the dependence function exploration and maximize redundant latch identification.

IV. Our Ordered Destroyed Cost Heuristic

In this section, Fig. 4 is used again to demonstrate the heuristic for dependent-latch identification in sequential circuits. At first, the dependence function network is used to determine which latch's dependence function exists with maximal input support candidates through the consideration of all other latches as input support candidates. However, its dependence function will not be derived immediately due to inaccuracy. Thus, two sets of latches $P : \{L_1, L_2, L_3, L_9, L_{10}\}$ and $I : \{L_4, \ldots, L_8\}$ are distinguished, where P is the set of possibly dependent latches in which functional dependences exist and I is the set of independent latches.

A. Refinement of the Set P by the Set I

The latches in I are the essential latches of the circuit and can be used as the base latches to replace the latches in P. We can determine whether the latches in $P : \{L_1, L_2, L_3, L_9, L_{10}\}$ depend solely on the latches in $I : \{L_4, \ldots, L_8\}$. In addition, constant latches in a circuit will also be identified in this step since constant latches can be presented as having functional dependences over any input support candidates. For example, to examine whether L_1 depends only on the set I or is a constant latch using the dependence function network, n is set to 5 in the constraint part of the network, and y_0 and $\langle y_1, \ldots, y_5 \rangle$ are regarded as the dependent latch L_1 and the base latches $\langle L_4, \ldots, L_8 \rangle$, respectively. The other latches in P can also be examined in the same manner. Fig. 4 shows that L_9 and L_{10} are the latches dependent on I, and their input support candidate sets are refined from $\{L_1, \ldots, L_8, L_{10}\}$ and $\{L_1, \ldots, L_9\}$ to $\{L_4, \ldots, L_8\}$. As a result, L_9 and L_{10} are put into the set R, which collects the dependent latches depending on I and the constant latches. Following this, set P is updated from $\{L_1, L_2, L_3, L_9, L_{10}\}$ to $\{L_1, L_2, L_3\}$.

B. Refinement of the Set P Using the Ordered Destroyed Cost

After the refinement by I, the latches in P either depend on some of the latches in P itself or some of the latches in P and I. That is, each latch L_i in P has a dependence function with input support candidates $P \setminus \{L_i\}$ or $(P \setminus \{L_i\}) \cup I$. Next, P is divided into two disjoint sets P_R and P_I. P_R and P_I are both subsets of P, and P_R is a set of latches that collects the latch which depends on the latches in $P_I \cup I$. For a latch in P, if it is not in P_I, it is in P_R. The goal is to maximize the P_R set and minimize the P_I set, because the more latches there are in P_R, the more latches can be identified as dependent latches.

Definition 3: For the latches in set P where P is the set of possibly dependent latches, if a latch L is removed from the input support candidates of the other latches in P, the number of functional dependences that would be destroyed is defined as the destroyed cost of L.

An ordered destroyed cost is proposed as the guide for the heuristic to move latches from P to P_I. Each latch in set P has a destroyed cost. The method for evaluating the destroyed cost of a latch L_i in set P, denoted as dc_i, is to count the number of functional dependences of other latches in P that would be destroyed by removing L_i from their input support candidates. If the functional dependence of L_j is destroyed by L_i, it means that L_j's functional dependence does not exist if L_i is removed from L_j's input support candidates. In other words, for L_j to be selected as a dependent latch, L_i must be a base latch for it.

```
EvaluateDestroyedCost( )
 Input : set P, set I
 Output : dc_i of each L_i in P
  for each latch L_i in P
   for all other latches L_j(j ≠ i) in P{
     Check existence of FD_j with input candidates
      (P \ {L_i, L_j}) ∪ I ;
     if (remove L_i destroys FD_j)
          dc_i++ ;
   }
 return dc_i of each L_i in P ;
```

* FD_j is the functional dependency of latch L_j

Fig. 6. Pseudocode of EvaluateDestroyedCost function.

With the destroyed cost of all latches in P, latches with higher destroyed cost will be dealt with first, and set P will finally be partitioned into sets P_R and P_I. According to the descending order of the destroyed cost, the L_i with the highest dc_i is removed to P_I, and we determine whether the remaining latches in P depend solely on the updated $P_I \cup I$. If a latch in P satisfies this dependence condition, it can be moved to P_R. If P is not empty, the L_i with the next highest dc_i can be further moved to P_I, and the aforementioned operations can be iterated. The same example in Section IV-A is used to demonstrate how to evaluate and use the destroyed cost. The physical meaning of the destroyed cost will be discussed at the end of this section. The pseudocode of evaluating the destroyed cost of latches in P is shown in Fig. 6.

To evaluate the destroyed cost of L_3, denoted as dc_3, L_3 is removed from the input support candidates of latches L_1 and L_2, and the dependence function network is used to examine how many functional dependences would not exist without L_3. That is, we count the number of L_j's functional dependences with candidates $(P \setminus \{L_3, L_j\}) \cup I$ that are destroyed by removing L_3. In this example, both L_1's functional dependence with input support candidates $\{L_2, L_4, \ldots, L_8\}$ and L_2's functional dependence with input support candidates $\{L_1, L_4, \ldots, L_8\}$ would be destroyed without L_3, according to the dependence function network. This results in a destroyed cost of $L_3 = 2$.

After all calculations, we obtain the destroyed cost $dc_1 = 1$, $dc_2 = 1$, and $dc_3 = 2$. According to the descending order of the destroyed cost, L_3 with $dc_3 = 2$ is first moved to P_I, and the remaining latches $\{L_1, L_2\}$ in P are checked to see if they functionally depend on the updated $P_I \cup I$ (updated $P_I : \{L_3\}$) using the dependence function network. In this iteration, $\{L_1, L_2\}$ have dependence functions with input support candidates $P_I \cup I$. Thus, they are moved to P_R. Finally, P is empty, and the heuristic has partitioned P into $P_I : \{L_3\}$ and $P_R : \{L_1, L_2\}$. All latches in P_R functionally depend on $P_I \cup I$.

Essential latch sets $P_I : \{L_3\}$, $I : \{L_4, L_5, L_6, L_7, L_8\}$
Dependent latch sets $P_R : \{L_1, L_2\}$, $R : \{L_9, L_{10}\}$

Corresponding input support candidates:

L_1's input support candidates = $P_I \cup I$
L_2's input support candidates = $P_I \cup I$
L_9's input support candidates = I
L_{10}'s input support candidates = I

Corresponding dependency functions:

$L_1 = L_3 \bullet \neg L_5$
$L_2 = L_3 \bullet \neg L_4$
$L_9 = L_6 + L_7 + L_8$
$L_{10} = \neg(L_6 + L_7 + L_8)$

Fig. 7. Combinational dependent latches identified by this heuristic.

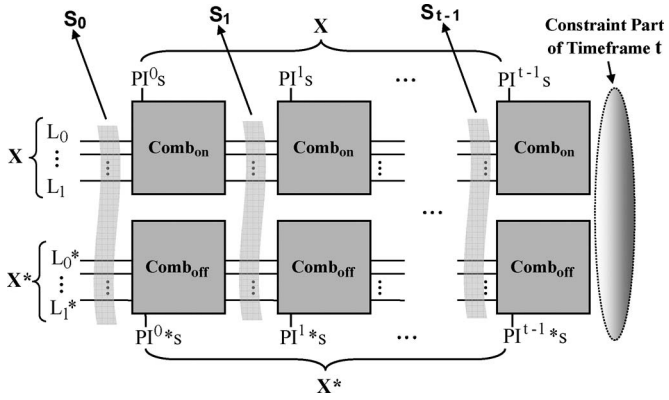

Fig. 8. Multi-time-frame dependence function network for sequential functional dependences.

Fig. 7 shows the results of this example using the proposed heuristic. In this example, the approach can identify four latches $(P_R : \{L_1, L_2\}, R : \{L_9, L_{10}\})$ as the combinational dependent latches by selecting $(P_I : \{L_3\}, I : \{L_4, \ldots, L_8\})$ as the essential (base) latches. In addition, the dependence functions of dependent latches can be explored with more accurate input support candidates as compared with Fig. 5 used in [17]. According to Theorem 4, the Boolean formula of each dependence function can also be derived, as shown in the bottom of Fig. 7, by this heuristic.

Next, the meaning of the ordered destroyed cost is explained. Latches with similar behavior would have the same destroyed cost. Therefore, grouping the latches by their destroyed costs and consecutively moving L_i from the same group to P_I until the group becomes empty speeds up the identification process. Furthermore, dealing with groups with a higher destroyed cost first might help to identify more dependent latches, since latches in the higher cost group might have a higher probability of replacing more dependent latches. In this example, if L_3 $(dc_3 = 2)$ is moved to P_I, it can replace two latches L_1 and L_2 with the independent latches $\{L_4, L_5\}$. However, if L_1 $(dc_1 = 1)$ is moved to P_I, only one latch could be replaced—L_2 or L_3—since the exact input support sets of L_2 and L_3 are $L_2 = \{L_3, L_4\}$ and $L_3 = \{L_1, L_2\}$, respectively. This example reveals that the order in which latches are moved from P to P_I influences the results obtained. In Section V, the experimental results of using different orderings will be presented. These results strongly support the proposed ordering.

C. Sequential Functional Dependence in Reachable State Space

To find more functional dependences existing in the reachable state space but not in the whole state space, a dependence function network with t time frames is proposed, as shown in Fig. 8. To build this dependence function network, $(0, \ldots, t-1)$ circuit parts of the network in Fig. 1 are expanded by wiring internal PPOs and PPIs and then connecting the same constraint part as that in Fig. 1 at the end of the network. This network can detect dependence for multiple time frames. In this paper, this functional dependence across multiple time frames is called the *sequential functional dependence*.

Definition 4: A functional dependence existing in the whole state space is called a combinational functional dependence. Functional dependences existing in the reachable state space but not in the whole state space are named sequential functional dependences.

The *sequential dependence* defined in [14] is literally similar to the sequential functional dependence defined in this paper, but actually, the two are very different. The BDD-based approach to sequential dependence in [14] cannot extract the functional dependence results from the transition functions at a time due to BDD size limitations. Therefore, two heuristics are proposed to collect new functional dependences iteratively. At each iteration, the new functional dependences explored are called sequential dependences. After all iterations, the obtained functional dependence results are equivalent to the combinational functional dependences of this work. However, the sequential functional dependences in this work are identified by unfolding time frames.

To find the functional dependence at time frame t, the dependence function network is first constructed as shown in Fig. 8. Then, the same method mentioned in Section II is used to detect the dependences in the constraint part of Fig. 8. In addition, given that a sequential system begins with an unrestricted initial state S_0, the reachable state space at each time frame is monotonically reduced, and a fixed point of reachable state space is reached. That is, $S_j \subseteq S_i$ if $j > i$ (S_t is the reachable state space at the tth time frame). Therefore, with the property of state-space-size reduction, if the functional dependence of L_i does not exist at time frame t, it might exist at time frame k $(k > t)$. However, the functional dependence of L_i found at time frame t must still hold at the succeeding time frame k $(k > t)$.

This approach will feed $P_I \cup I$, which is found at the current time frame, to the next time-frame network to get more dependences until the run-time limit is reached. That is, at time frame t, all the latches that are to be dealt with are only the latches in $P_I \cup I$ at time frame $(t-1)$. Therefore, if latches in $P_I \cup I$ identified at time frame $(t-1)$ have functional dependences at time frame t, some additional dependent latches can be identified with the same heuristic, ordered destroyed cost. If that is found, the previous dependence functions will be updated. Before feeding $P_I \cup I$ to the next time-frame network, a refined process is conducted to examine whether latches in P_I are actually used by a dependent latch's dependence function at the

Dependent latches: $\{L_1, L_2, L_9, L_{10}\}$ after timeframe 1

Corresponding dependency functions:

$L_1 = L_3 \bullet \neg L_5$
$L_2 = L_3 \bullet \neg L_4$
$L_9 = L_6 + L_7 + L_8$
$L_{10} = \neg(L_6 + L_7 + L_8)$

Dependent latches: $\{L_1, L_2, L_6, L_9, L_{10}\}$ after timeframe 2

Corresponding dependency functions:

$L_1 = L_3 \bullet \neg L_5$
$L_2 = L_3 \bullet \neg L_4$
$L_6 = \neg L_7 \bullet L_8$
$L_9 = L_7 + L_8$
$L_{10} = \neg(L_7 + L_8)$

Fig. 9. Dependent latches identified at different time frames by this approach.

current time frame. Since the heuristic might not obtain the optimal results every time, this process is used to refine the results. If some latches in P_I are not used, they will be removed from P_I to P, and the aforementioned operations will be iterated. For the last example, $P_I \cup I = \{L_3\} \cup \{L_4, L_5, L_6, L_7, L_8\}$ is fed into the next time-frame network. Since L_3 in P_I is actually used by the derived dependence functions of L_1 and L_2, the refined process will be terminated. At the next time frame, a new sequential dependence function $(L_6 = \neg L_7 \bullet L_8)$ is found. Hence, there is another dependent latch L_6 after time frame 2. The previous dependent latches' dependence functions, which depend on L_6, must also be updated. In this example, the dependence function of L_9 is updated from $(L_9 = L_6 + L_7 + L_8)$ to $(L_9 = \neg L_7 \bullet L_8 + L_7 + L_8 = L_7 + L_8)$, and the dependence function of L_{10} is updated from $(L_{10} = \neg(L_6 + L_7 + L_8))$ to $(L_{10} = \neg(\neg L_7 \bullet L_8 + L_7 + L_8) = \neg(L_7 + L_8))$ if L_6 is considered as a dependent latch after time frame 2. The combinational and sequential dependent latches of this example identified by this approach are shown in Fig. 9.

The following paragraphs will explain an accelerated technique used in sequential-dependent-latch identification and will introduce an application of the identified sequential functional dependences.

1) Accelerated Technique for Identification of Sequential Dependent Latches: The accelerated technique for identifying sequential dependent latches consists of two parts, the incremental technique [28] of SAT solvers and the early detection of dependent latches. To find the sequential functional dependence at time frame t using SAT solvers, it needs to set the variables in the CNFs of circuit parts belonging to time frame $(0, \ldots, t-1)$. Since these variables have been processed at time frame $(0, \ldots, t-1)$, much useful information is available to speed up the process at time frame t. Thus, the heuristic uses the incremental technique of SAT solvers to reuse the learned clauses at time frame $(0, \ldots, t-1)$. Furthermore, dependent latches existing at time frame t' $(t' \geq 1)$ must also exist at time frame t $(t > t')$. That is, sequential dependent latches existing at time frame t $(t > 1)$ may have already existed at time frame t' $(1 \leq t' \leq t-1)$. Therefore, these dependent latches detected earlier at time frame t' $(1 \leq t' \leq t-1)$ are recorded, and the run time for identifying all sequential dependent latches at time frame t $(t > 1)$ can be reduced a lot. In other words, to identify dependent latches at time frame t, our approach seeks to identify dependent latches from time frame 1 to t, and only latches in $P_I \cup I$ at the current time frame are needed for the next time frame.

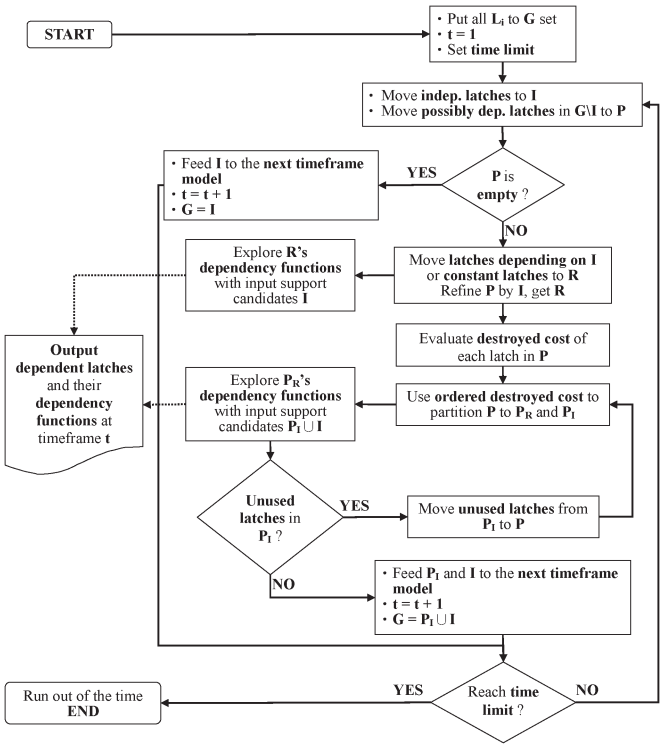

Fig. 10. Flowchart of the proposed algorithm.

2) Application of Sequential Functional Dependence: This heuristic can find the combinational functional dependence existing in the whole state space, as well as the sequential functional dependence only existing in the reachable state space. Therefore, an application of this work is to reduce the efforts of reachability analysis. With information about the sequential functional dependence at each time frame, the reachability analysis engine that traverses the reachable state space of a sequential circuit can ignore some dependent latches after a certain time frame. Hence, the search space is significantly reduced. Furthermore, for a sequential circuit that allows t don't-care initializing cycles to run from unrestricted initial states before entering the normal operation states, only the state space in time frame t is considered (also known as delay replaceability in [26]). Thus, the sequential functional dependences found at time frame t could be used to further optimize the design under this situation.

D. Overall Algorithm

The complete flowchart of this approach is shown in Fig. 10. At first, all latches of a circuit are put into a global set G, and the time-frame variable t is set to 1 to identify dependent latches at time frame 1 with the unrestricted initial states. Considering all other latches in G as input support candidates to examine the existence of each latch's dependence function, G can be partitioned into the possibly dependent latch set P and the independent latch set I. If P is empty, it means that no

TABLE I
COMBINATIONAL-DEPENDENT-LATCH IDENTIFICATION FOR STANDARD BENCHMARKS IN ALL STATE SPACES

Circuit	|Node|	|FF|	|Dep. latch|	|Ess. latch|	Dep. latch	Time (S)
			$R + P_R$	$I + P_I$	ident. ratio (%)	
s5378	2779	164	$1 + 7$	$135 + 21$	4.88	2.08
s9234	5597	211	$1 + 19$	$165 + 26$	9.48	9.86
s15850	9786	597	$8 + 14$	$568 + 7$	3.69	11.27
s13207	8027	669	$5 + 154$	$453 + 57$	23.77	283.18
s38584	19407	1452	$2 + 11$	$1428 + 11$	0.90	72.89
s38417	22397	1636	$0 + 71$	$1541 + 24$	4.34	312.07
s35932	16353	1728	$0 + 0$	$1728 + 0$	0	67.40
b12	952	121	$0 + 2$	$117 + 2$	1.65	0.11
b14	9821	245	$2 + 0$	$243 + 0$	0.82	3.65
b15	8437	449	$0 + 0$	$449 + 0$	0	5.27
b21	20049	490	$4 + 0$	$486 + 0$	0.82	21.15
b22	29184	735	$6 + 0$	$729 + 0$	0.82	67.74

TABLE II
COMBINATIONAL-DEPENDENT-LATCH IDENTIFICATION FOR RETIMED BENCHMARKS IN ALL STATE SPACES

Circuit (retimed)	|Node|	|FF|	|Dep. latch|	|Ess. latch|	Dep. latch	Time (S)
			$R + P_R$	$I + P_I$	ident. ratio (%)	
s5378	2887	366	$11 + 136$	$139 + 80$	40.16	131.17
s9234	3393	373	$7 + 140$	$187 + 39$	39.41	70.70
s15850	10069	1708	$70 + 495$	$947 + 196$	33.08	3600.56
s13207	8144	1533	$6 + 548$	$814 + 165$	36.14	3552.05
s38584	19864	4796	$163 + 2646$	$1188 + 799$	58.57	284594.95
s38417	22272	3894	$190 + 1412$	$1914 + 378$	41.14	46732.26
s35932	23644	9316	$288 + 297$	$7858 + 873$	6.28	54754.45
b12	3667	2877	$0 + 41$	$2797 + 39$	1.43	39.87
b14	16880	8145	$5 + 707$	$7324 + 109$	8.74	7552.50
b15	8610	1240	$12 + 450$	$533 + 245$	37.26	31732.38
b21	23127	5415	$5 + 1421$	$3714 + 275$	26.33	60798.93
b22	29173	3147	$21 + 1729$	$1085 + 312$	55.61	105773.51

functional dependence exists at time frame t. Therefore, G is then updated with I at the current time frame and proceeds to the next time-frame iteration. Otherwise, P is first refined by using I, and we obtain the latch set R where latches depend only on I or have constant values. The dependence functions of latches in R with the input support candidate set I are also explored. Next, the ordered destroyed cost are evaluated and used to partition P into P_R and P_I, and the latches' dependence functions in P_R with the input support candidate set $P_I \cup I$ are also explored. After exploring the dependence functions of P_R, it is established whether any latch in P_I has not been used by P_R's dependence functions. Any unused latch is then removed from P_I to P, and the partition is refined again based on the ordered destroyed cost. When all latches in P_I are actually used by the dependent latches in P_R, dependent latches in $R \cup P_R$ are returned along with their dependence functions. G is then updated with $P_I \cup I$ at the current time frame, and the next time-frame iteration proceeds. The algorithm terminates after reaching a predefined time limit.

V. EXPERIMENTAL RESULTS

The proposed heuristic was implemented within SIS [25] environment, and MiniSAT [9] was used as the SAT solver. The experiments were conducted on a Linux platform (CentOS 4.4) with a 2.194-GHz machine and 8 GB of memory. To show the scalability of the algorithm, a set of larger sequential circuits [more flip-flops (FFs)] were selected from ISCAS'89 and ITC'99 benchmarks.

The experimental results are shown in two sections. Section V-A shows our approach's capability of identifying dependent latches. Section V-B shows how dependent-latch identification quantitatively improves reachability analysis.

A. Dependent-Latch Identification Using the Proposed Approach

This section provides the experimental results of the proposed approach in dependent-latch identification.

1) Combinational-Dependent-Latch Identification: Table I shows the results of combinational dependent latches identified by this approach. Column 1 lists the benchmarks. Columns 2 and 3 list the number of nodes and the number of FFs (also called latches), respectively. The combinational-dependent-latch results of the proposed approach are listed in Columns 4–7. |Dep. latch| represents the number of identified dependent latches. |Ess. latch| represents the number of essential latches which can be considered as base latches to replace the dependent latches. The next two columns show the percentage of FFs that are identified as dependent latches (i.e., |Dep. latch|/|FF|) and the CPU time measured in seconds.

For example, the s15850 circuit has 9786 nodes and 597 FFs. This approach can identify 22 $(8 + 14)$ dependent latches and 575 $(568 + 7)$ essential latches within 11.27 s. Therefore, 3.69% of latches are identified as combinational dependent latches by the proposed approach.

In this experiment, we observed that the proposed approach identifies more dependent latches with the ISCAS benchmarks than with the ITC benchmarks. This indicates that fewer dependent latches exist in the ITC benchmarks with the unrestricted initial states. Therefore, the results of the retimed version of the same benchmarks, which may have more dependent latches, are shown in the next experiment to demonstrate the capabilities of the proposed approach.

The experimental results of combinational dependent latches identified by this approach for retimed benchmarks are shown in Table II. Column 1 lists the benchmarks after the retiming (using ABC [6] command, "most forward retiming"). For example, the retimed version of the s13207 circuit has 8144 nodes and 1533 FFs, in which this approach can identify 554 $(6 + 548)$ dependent latches and 979 $(814 + 165)$ essential latches. Therefore, in the retimed version of the s13207 circuit, 36.14% of the latches in the circuit are identified as combinational dependent latches. As demonstrated in Table II, this approach can recognize a large number of dependent latches, so that fewer latches have to be considered in reachability analysis. However, the time expenditure may be a concern in the dependent-latch identification of benchmarks which have a high ratio of dependent latches. We observed that the run time of the proposed approach also increases for this version of benchmarks. This is because the run time of the approach strongly depends on the number of nodes and the number of dependent latches in it. Fortunately, compared with the

TABLE III
INFLUENCE OF ORDERED DESTROYED COST HEURISTIC ON DEPENDENT-LATCH IDENTIFICATION

Circuit	$\|P\|$	Descending	Ascending	Middle	Random	Ratio (Des. P_R / x P_R)		
		$P_R + P_I$	$P_R + P_I$	$P_R + P_I$	$P_R + P_I$	x = Asc.	x = Mid.	x = Rand.
s5378	28	$7 + 21$	$6 + 22$	$7 + 21$	$6.4 + 21.6$	1.167	1	1.094
s9234	45	$19 + 26$	$19 + 26$	$19 + 26$	$19 + 26$	1	1	1
s15850	21	$14 + 7$	$14 + 7$	$14 + 7$	$14 + 7$	1	1	1
s13207	211	$139 + 72$	$138 + 73$	$138 + 73$	$142.8 + 68.2$	1.007	1.007	0.973
s38584	22	$11 + 11$	$11 + 11$	$11 + 11$	$11 + 11$	1	1	1
s38417	95	$70 + 25$	$62 + 33$	$62 + 33$	$64.2 + 30.8$	1.129	1.129	1.090
b12	4	$2 + 2$	$2 + 2$	$2 + 2$	$2 + 2$	1	1	1

TABLE IV
DEPENDENT-LATCH IDENTIFICATION WITHOUT OR WITH THE REFINED PROCESS

Circuit	$\|P\|$	DC. order	W/O. ref. process	W. ref. process	Ratio
			P_R + P_I	$P_R + P_I$	W. P_R / W/O. P_R
s5378	28	Des.	$7 + 21$	$7 + 21$	1
s5378	28	Asc.	$6 + 22$	$7 + 21$	1.17
s13207	211	Des.	$139 + 72$	$154 + 57$	1.11
s13207	211	Asc.	$138 + 73$	$154 + 57$	1.12
s38417	95	Des.	$70 + 25$	$71 + 24$	1.01
s38417	95	Asc.	$62 + 33$	$63 + 32$	1.02

time saved through reachability analysis due to dependent-latch identification, it is worth spending the extra run time on this component.

2) Distinct Orders of Destroyed Cost: For the experiment in Table I, the dependent-latch identification is performed in descending order of destroyed cost. For this section, we repeat the experiment using another three distinct orders of destroyed cost: ascending, middle, and random orders.

1) Ascending order: The latch L_i with the lowest dc_i in P is moved to P_I first.
2) Middle order: The latch L_i with a middle value dc_i in P is moved to P_I first.
3) Random order: A latch L_i in P is randomly moved to P_I.

The comparisons of the results among the experiments in these four orders are shown in Table III.

Column 1 lists the benchmarks that have nonempty P ($P_R + P_I$) in Table I. Column 2 lists the number of latches in P. The set P, which is divided into two disjoint sets P_R and P_I by our algorithm, with respect to the four orders of the destroyed cost are listed in Columns 3–6, respectively. The ratios of P_R obtained with the descending order and the other orders of the destroyed cost are shown in the final three columns. Note that the results of random order are averagely obtained by repeating the experiments ten times.

The experimental results show that the heuristic done using these four orders have the same results for some benchmarks. However, for s5378 and s38417, descending order is a better choice. For example, s38417 has 95 latches in P, and P is partitioned into P_R and P_I by the heuristic. The heuristic with the descending order of destroyed cost can identify 70 dependent latches in P_R, but merely 62–65 dependent latches in P_R are identified by the experiments done with the other three orders. For s13207, however, random order is a better choice. The reason that our heuristic obtained the same results with these four orders in some cases is as follows: For circuits that only have equivalence relation or circular functional dependences among latches, there is no difference among using any orders. Since the results with the descending order are often better than that of the other orders, we choose the descending order of destroyed cost as our heuristic.

3) Dependent-Latch Maximization With the Refined Process: This section demonstrates the results of running the additional refined PROCESS on P_R and P_I for maximizing the P_R set and minimizing the P_I set. Table IV shows the results of the combinational-dependent-latch identification with or without the refined process. Column 1 lists the benchmarks which have different results with or without the refined process. Column 2 shows their corresponding number of latches in P. Column 3 shows the order of the destroyed cost in the experiment. Des. represents the descending order, and Asc. represents the ascending order. Columns 4 and 5 list the results without and with the use of the refined process, respectively. The ratio of P_R obtained with the refined process and without the refined process is shown in the final column.

According to Table IV, the ratios of P_R in these benchmarks are greater or equal to 1, regardless of destroyed cost order. Therefore, the refined process is shown as an effective procedure to improve the results. Take s13207 with the descending order as an example: 15 more latches are recognized in the P_R set when we apply the refined process.

4) Sequential-Dependent-Latch Identification: The experimental results on the sequential functional dependence in the

TABLE V
SEQUENTIAL-DEPENDENT-LATCH IDENTIFICATION IN THE REACHABLE STATE SPACE FOR 10 000-S RUN-TIME LIMIT

Circuit	|FF|	All state space	Reachable state space			Unfolded	All dep.
		|Dep.|	|Dep.|	Timeframe	Ratio (%)	timeframe	ratio (%)
s5378	164	8	53	14	27.44	106	32.32
s9234	211	20	22	2	0.95	27	10.43
s15850	597	22	62	8	6.70	23	10.39
s13207	669	159	250	22	13.60	26	37.37
s38584	1452	13	32	4	1.31	7	2.20
s38417	1636	71	140	5	4.22	5	8.56
s35932	1728	0	0	0	0	9	0
b12	121	2	2	1	0	109	1.65
b14	245	2	2	1	0	27	0.82
b15	449	0	0	0	0	33	0
b21	490	4	4	1	0	13	0.82
b22	735	6	6	1	0	8	0.82

TABLE VI
CPU TIME COMPARISON FOR DEPENDENT-LATCH IDENTIFICATION IN S38417 WITH OR WITHOUT ACCELERATED TECHNIQUE

SAT	CPU time (S)					Dep. latch
	T1	T2	T3	T4	T5	ratio at T5
W. acce.	312.07	1,517.05	3,831.07	6,846.17	9,957.80	8.56%
W/O. acce.	303.47	3,525.69	18,144.63	73,298.95	163,803.95	7.40%

reachable state space are shown in Table V. The CPU time limit is set to 10 000 s. Column 3 lists the number of identified dependent latches considered in all state spaces (i.e., at time frame 1). The results of sequential functional dependence are listed in Columns 4–6. Column 4 shows the number of dependent latches identified after the time frame in Column 5. Column 6 shows the ratio of additionally identified sequential dependent latches as compared with the total FFs in Column 2 (i.e., $((|\text{Dep.}|\text{in Column}4) - (|\text{Dep.}|\text{in Column}3))/|\text{FF}|$). Column 7 lists the number of time frames unfolded and examined within the 10 000-s run-time limit. The ratio of the number of overall dependent latches identified after the time frame in Column 5 is shown in the final column.

The experimental results show that sequential functional dependences may exist in the circuit, which can be used for reachability analysis and sequential circuit optimization. Take s5378 as an example: The state space is reduced from 2^{164} to 2^{156} by ignoring the combinational dependent latches. Furthermore, the state space is shrunk from 2^{156} to 2^{111} by ignoring sequential dependent latches after time frame 14. However, from time frame 14 to 106, there are no additional sequential dependent latches that can be identified. The reasons for it are that the state space might have shrunk to a fixed point or some sequential dependent latches may exist after time frame 107. However, we cannot identify them within the time limit.

5) Implementation With the Accelerated Technique: The proposed accelerated technique, consisting of an incremental method and early detection, is used to enhance the performance of this approach. The performance comparison between the approaches with or without the accelerated technique while identifying dependent latches is also shown in Table VI, where a large sequential circuit s38417 is used as the benchmark.

Rows 1 and 2 show the time used for dependent-latch identification at each time frame with and without employing the accelerated technique, respectively. The approach without using the accelerated technique has to build a new SAT instance and deal with all latches in a benchmark at each time frame. The final column lists the percentage of identified dependent latches for these two approaches at the fifth time frame, T5.

According to Table VI, except T1, the CPU time can be significantly reduced with the proposed accelerated technique. It saves 56.97%, 78.89%, 90.66%, and 93.92% of CPU time at T2, T3, T4, and T5, respectively. The reason for the time saving is that the clauses learned at time frame $0 \sim (t-1)$ could be reused in the multi-time-frame network of time frame t, i.e., these clauses could reduce the search space at time frame t as in the concept of branch-and-bound algorithms. Another reason is that this approach only delivers the $P_I \cup I$ latch set obtained at the current time frame to the next time frame. Hence, the network in the next time frame focuses only on the latches in $P_I \cup I$, rather than all latches. With this additional information, the approach with the accelerated technique identifies 1.16% more dependent latches $(8.56\% - 7.40\% = 1.16\%)$ than that not using this technique.

B. Reachability Analysis Enhancement With the Dependent-Latch Identification

This section provides the experimental results of the reachability analysis with and without using the results of the dependent-latch identification within VIS [3] environment, the state-of-the-art academic formal verification tool. The sequential circuits in ISCAS'89 which have dependent latches identified in Section V-A are selected as the benchmarks.

TABLE VII
EXPERIMENTAL RESULTS OF REACHABILITY ANALYSIS FOR REACHING THE SAME TIME-FRAME WITHIN 60 000-S RUN-TIME LIMIT IN THE ORIGINAL AND THE OPTIMIZED CIRCUITS OF THE STANDARD ISCAS'89 BENCHMARKS

Circuit	Reached states		Ori.			Opt.			Reduction ratio $(Ori. - Opt.)/Ori.$	
	Timeframe	|RState|	|FF|	BDD size	Time (S)	|FF|	BDD size	Time (S)	BDD size (%)	Time (%)
s5378	5	$1.24393e+16$	164	1,653,300	57,248	156	270,723	49,869	83.63	12.89
s9234	11	$1.65102e+16$	211	942,896	31,882	191	605,665	16,861	35.77	47.11
s15850	16	$1.52765e+07$	597	1,530,444	30,598	575	1,393,033	23,751	8.98	22.38
s13207	38	$1.44021e+15$	669	4,913,692	48,491	510	1,439,756	12,935	70.70	73.32
s38584	11	$8.86230e+08$	1,452	8,818,246	19,683	1,439	8,145,184	41,420	7.63	−110.44
s38417	3	$3.45877e+18$	1,636	315,465	8,550	1,565	218,134	19,600	30.85	−129.24

Note: BDD size is reported from VIS.

TABLE VIII
EXPERIMENTAL RESULTS OF REACHABILITY ANALYSIS WITHIN A RUN-TIME LIMIT IN THE ORIGINAL AND THE OPTIMIZED CIRCUITS OF THE STANDARD ISCAS'89 BENCHMARKS

Circuit	Time limit (S)	Ori.				Opt.				|RState| improvement (Opt. / Ori.) (%)	BDD size reduction (%)
		|FF|	Timeframe	|RState|	BDD size	|FF|	Timeframe	|RState|	BDD size		
s5378	300,000	164	6	$6.77711e+16$	1,564,629	156	7	$1.63316e+17$	452,403	240.98	71.09
s9234	300,000	211	12	$4.64715e+17$	1,651,445	191	12	$4.64715e+17$	1,221,122	100	26.10
s15850	600,000	597	18	$7.21485e+07$	5,897,688	575	18	$7.21485e+07$	4,383,715	100	25.67
s13207	600,000	669	43	$4.82309e+15$	8,301,281	510	60	$1.44247e+17$	10,762,172	2990.76	−29.64
s38584	900,000	1,452	13	$1.61008e+10$	35,855,891	1,439	13	$1.61008e+10$	23,424,952	100	34.67
s38417	900,000	1,636	3	$3.45877e+18$	315,465	1,565	3	$3.45877e+18$	218,134	100	30.85

Note: BDD size is reported from VIS.

Although both the combinational and sequential dependent latches in a sequential circuit can be identified by our approach, this experiment only considers the combinational ones since VIS does not allow setting the sequential dependent latches during the reachability analysis. The optimized version of one benchmark is the one that completely removes out the combinational dependent latches of the original circuit and reconnects the circuit based on the corresponding dependence functions. The reachability analysis are performed on these two versions of circuits. The initial states of all latches in the optimized circuits are set to 0. However, for the original circuits, only the essential latches are set to 0. The dependent latches in the original circuits are set to the values with respect to the dependence functions under the essential latches' setting. The dynamic variable ordering technique in BDD, sift algorithm, is used in the experiments (VIS command, "dynamic_var_ordering -e sift"). To clearly demonstrate the improvement of reachability analysis with the identified combinational dependent latches, two sets of benchmarks are used. One is the standard ISCAS'89 benchmark, and the other is the retimed ISCAS'89 benchmark. Furthermore, we also conduct two experiments with different terminating conditions, one is time-frame limit and the other is run-time limit, and compare the results in terms of the number of reached states, CPU time, and BDD size, respectively. Note that the time for computing the latch dependence is excluded. The following sections present detailed experimental descriptions and result comparisons.

1) Standard ISCAS'89 Benchmark:

a) Time-frame limit: Table VII shows the experimental results of the reachability analysis done on the original and the optimized circuits with the same time-frame limit. Columns 2 and 3 list the reached time-frame limit and the number of reached states in this limited time frame within a 60 000-s run time. The results of the original and optimized benchmarks are listed in Columns 4–9. |FF| represents the number of FFs. The next two columns show the BDD size and the CPU time needed at the time frame. Columns 10 and 11 list the reduction ratio of the optimized circuit to the original one in BDD size and run time.

Take s5378 as an example: The original circuit has 164 FFs, and the optimized circuit has 156 FFs through the removal of eight combinational dependent latches $(R + P_R = 1 + 7)$. While both reached the same number of states $(1.24393e + 16)$ at the fifth time frame, the original circuit built a BDD with size 1 653 300 and required 57 248 s of run time, while the optimized circuit needed a 270 723 BDD size and 49 869 s of run time. The reduction ratio of BDD size and run time are 83.63% and 12.89%, respectively. For s38584 and s38417, however, the run time for the reachability analysis of the optimized circuits is more than that of the original circuits, although the optimized circuit still saves 7.63% and 30.85% in BDD size, respectively. The reason for the longer run time could be the dynamic ordering heuristic in VIS. This heuristic is triggered automatically and, unfortunately, is an uncertainty in the experiments. It is time consuming but does not always result in significant BDD size reduction. Nevertheless, we still apply this technique in the experiments since BDD size is the key factor for reaching deeper time frames.

b) Run-time limit: Table VIII shows the experimental results of the reachability analysis of the original and the optimized circuits with the same run-time limit. Columns 1 and 2 list the benchmarks and the corresponding run-time limit. Longer run-time limits are for more complicated circuits. The results of the original and the optimized benchmarks are listed

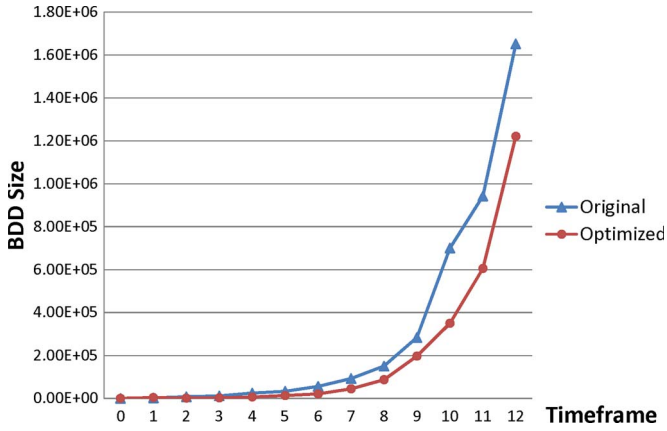

Fig. 11. BDD size of the original and optimized circuit during the reachability analysis for the standard s9234 benchmark.

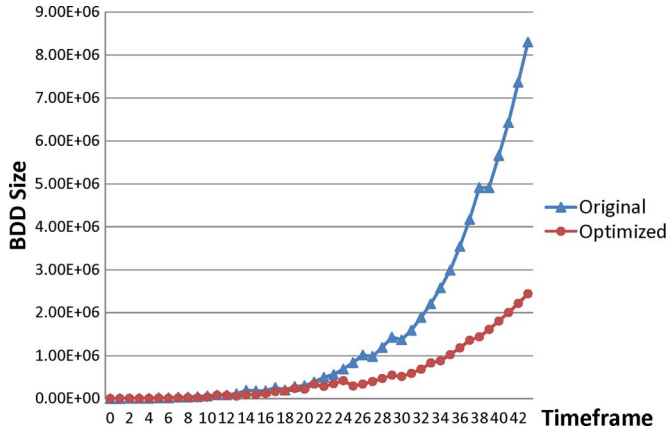

Fig. 12. BDD size of the original and optimized circuit during the reachability analysis for the standard s13207 benchmark.

in Columns 3–10. The Timeframe column shows the time frame reached within the run-time limit. The number of reached states and the corresponding BDD size are listed in the next two columns. Column 11 lists the improvement ratio of the optimized circuit to the original one in terms of the number of reached states. The final column lists the ratio of the BDD size reduction.

Take s13207 as an example; the original circuit has 669 FFs, while the optimized circuit has 510 FFs after removing 159 combinational dependent latches $(R + P_R = 5 + 154)$. Within the 600 000-s run-time limit, the original circuit reaches $4.82309e + 15$ states in the 43rd time frame while the optimized circuit reaches $1.44247e + 17$ states in the 60th time frame. The fewer FFs in the optimized circuits allowed 17 additional time frames to proceed and 2890.76% $(2990.76 - 100)$ more states to be reached with a 29.64% BDD size overhead. In addition, the optimized s5378 circuit also reached 140.98% $(240.98 - 100)$ more states than the original one and had a 71.09% BDD size reduction. For the circuits with the same state numbers—s9234, s15850, s38584, and s38417—the optimized versions still saved 26.10%, 25.67%, 34.67%, and 30.85% of BDD size, respectively. The BDD size savings allow the reachability analysis to possibly proceed further if the time limit is loosened.

The detailed analyses on BDD size at each time frame of the reachability analysis for the original circuits and the optimized ones are shown in Figs. 11–14 for circuits reached beyond ten time frames.

According to Figs. 11 and 12, the required BDD size for the reachability analysis of the optimized circuit is smaller than that of the original circuit in s9234 and s13207. In addition, the curves from s15850 and s38584 in Figs. 13 and 14 intersect at later time frames, which indicates that the optimized circuits potentially consume less memory and possibly proceed to deeper time frames.

2) Retimed ISCAS'89 Benchmark: The experiments in this section are the same with that in Section V-B-1, but for the retimed ISCAS'89 benchmark. The experimental results with the time-frame and run-time limits are summarized in Tables IX and X, respectively. Take s13207 in Table IX as an example; the optimized circuit saves 91.87% BDD size and 96.67% run time when reaching the same time frame as the original circuit. From Table X, the optimized s13207 reaches more than two orders of

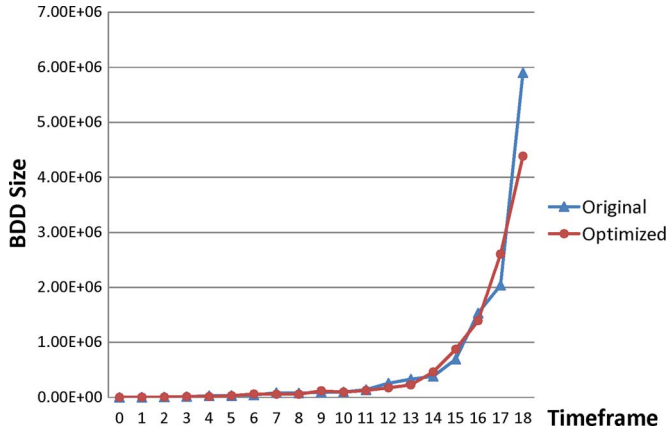

Fig. 13. BDD size of the original and optimized circuit during the reachability analysis for the standard s15850 benchmark.

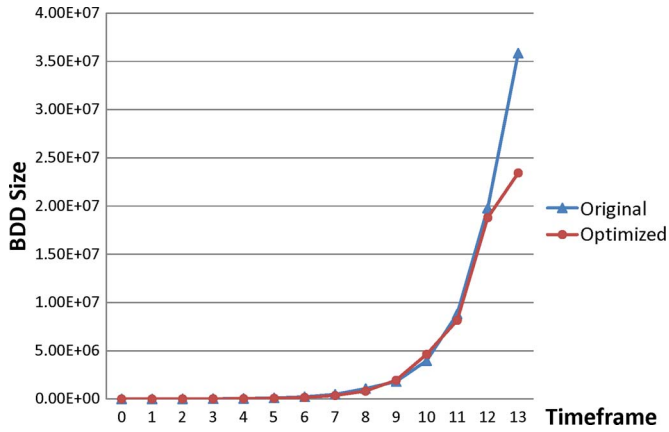

Fig. 14. BDD size of the original and optimized circuit during the reachability analysis for the standard s38584 benchmark.

magnitude states than that of the original one within the same run-time limit. Furthermore, the BDD size also has a 38.14% reduction. These results strongly support our heuristic of identifying much more dependent latches. Figs. 15–17 also show the BDD size comparison between the original and optimized circuits at each time frame for the retimed circuits reached beyond ten time frames. They reveal the same message that optimized circuits consume less memory for BDD construction and could proceed to deeper time frames.

TABLE IX
EXPERIMENTAL RESULTS OF REACHABILITY ANALYSIS FOR REACHING THE SAME TIME-FRAME WITHIN 60 000-S RUN-TIME LIMIT IN THE ORIGINAL AND THE OPTIMIZED CIRCUITS OF THE RETIMED ISCAS'89 BENCHMARKS

Circuit (retimed)	Reached states		Ori.			Opt.			Reduction ratio $(Ori. - Opt.)/Ori.$	
	Timeframe	|RState|	|FF|	BDD size	Time (S)	|FF|	BDD size	Time (S)	BDD size (%)	Time (%)
s5378	8	$9.95202e+15$	366	1,716,904	53,816	219	1,129,579	5,344	34.21	90.07
s9234	10	$1.23052e+11$	373	1,046,019	43,976	226	664,135	2,575	36.51	94.14
s15850	15	$3.43354e+06$	1708	5,464,881	58,913	1143	2,283,437	38,415	58.22	34.79
s13207	12	$3.29990e+07$	1533	5,579,199	57,272	979	453,332	1,907	91.87	96.67

Note: BDD size is reported from VIS.

TABLE X
EXPERIMENTAL RESULTS OF REACHABILITY ANALYSIS WITHIN A RUN-TIME LIMIT IN THE ORIGINAL AND THE OPTIMIZED CIRCUITS OF THE RETIMED ISCAS'89 BENCHMARKS

Circuit (retimed)	Time limit (S)	Ori.				Opt.				|RState| improvement (Opt. / Ori.) (%)	BDD size reduction (%)
		|FF|	Timeframe	|RState|	BDD size	|FF|	Timeframe	|RState|	BDD size		
s5378	600,000	366	9	$2.48322e+16$	2,051,308	219	16	$7.47998e+16$	4,546,136	301.22	−121.62
s9234	600,000	373	11	$9.51771e+11$	2,376,219	226	12	$1.80544e+13$	2,498,533	1896.93	−5.15
s15850	900,000	1708	17	$1.63871e+07$	14,069,944	1143	17	$1.63871e+07$	5,707,679	100	59.43
s13207	900,000	1533	37	$4.54249e+11$	21,581,617	979	65	$5.63007e+13$	13,351,440	12394.24	38.14

Note: BDD size is reported from VIS.

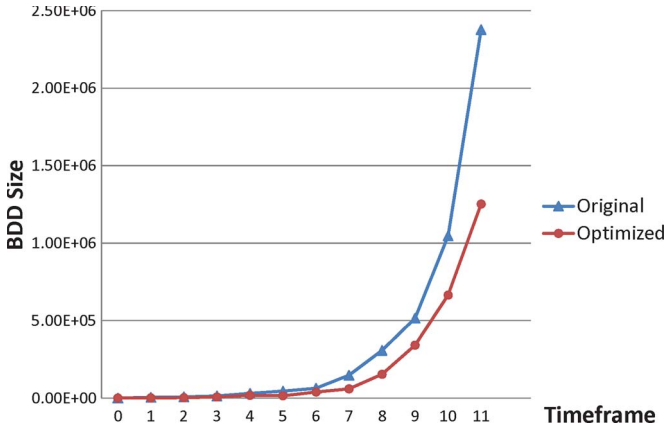

Fig. 15. BDD size of the original and optimized circuit during the reachability analysis for the retimed s9234 benchmark.

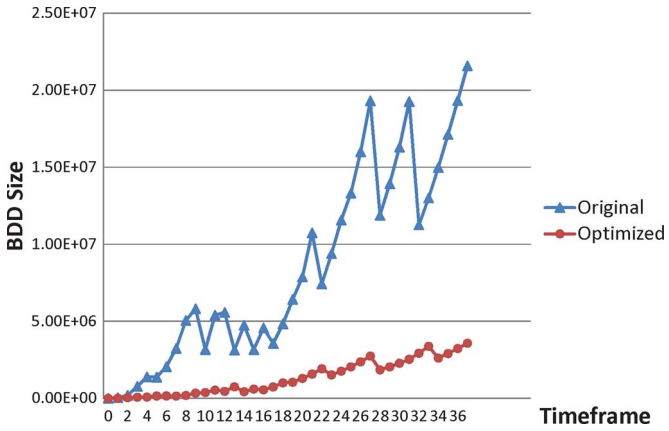

Fig. 16. BDD size of the original and optimized circuit during the reachability analysis for the retimed s13207 benchmark.

VI. CONCLUSION

This paper has proposed an algorithm to efficiently and precisely identify combinational dependent latches and explore their dependence functions with the more accurate input support candidates. Furthermore, identifying dependent latches without exploring dependence functions first could also benefit the identification results. In addition, a new multi-timeframe dependence function network has been proposed to identify additional sequential dependent latches existing in the reachable state space. With the information about sequential dependent latches, the state space of sequential circuits can be further reduced, and the circuits can be further optimized. With the dependent latches identified, the complexity of performing reachability analysis can be reduced by disregarding these dependent latches as seen in the presented experimental results.

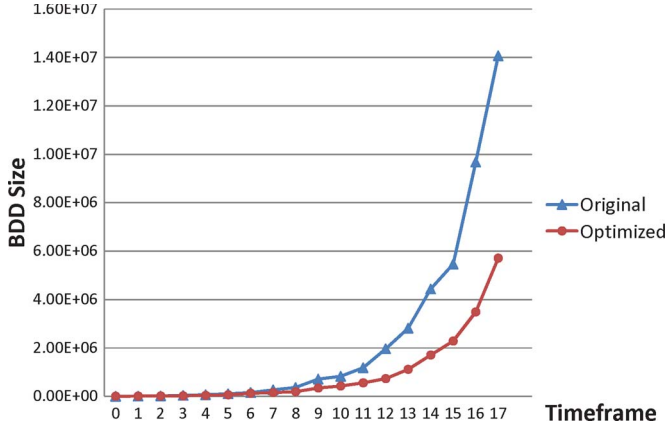

Fig. 17. BDD size of the original and optimized circuit during the reachability analysis for the retimed s15850 benchmark.

REFERENCES

[1] F. A. Aloul, I. L. Markov, and K. A. Sakallah, "Faster SAT and smaller BDDs via common function structure," Univ. Michigan, Ann Arbor, MI, Dec. 12, 2001. Tech. Rep.
[2] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[3] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa, "VIS: A system for verification and synthesis," in *Proc. Comput.-Aided Verification Conf.*, 1996, pp. 423–427.
[4] D. Brand, "Verification of large synthesized designs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 1993, pp. 534–537.
[5] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill, "Symbolic model checking for sequential circuit verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 4, pp. 401–424, Apr. 1994.
[6] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*. Release 70930. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/
[7] W. Craig, "Linear reasoning: A new form of the Herbrand–Gentzen theorem," *J. Symb. Log.*, vol. 22, no. 3, pp. 250–268, 1957.
[8] H. Cho, G. D. Hatchel, E. Macii, B. Plessier, and F. Somenzi, "Algorithms for approximate FSM traversal based on state space decomposition," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1993, pp. 25–30.
[9] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.*, 2003, pp. 502–518.
[10] S. G. Govindaraju, D. L. Dill, A. Hu, and M. A. Horowitz, "Approximate reachability analysis with BDDs using overlapping projections," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1998, pp. 451–456.
[11] A.-J. Hu and D. L. Dill, "Reducing BDD size by exploiting functional dependencies," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1993, pp. 266–271.
[12] S.-Y. Huang, K.-T. Cheng, and K.-C. Chen, "AQUILA: An equivalence verifier for large sequential circuits," in *Proc. Asian South Pacific Des. Autom. Conf.*, 1997, pp. 455–460.
[13] R. Hojati, S. Krishnan, and R. K. Brayton, "Heuristic algorithms for early quantification and partial product minimization," UC Berkeley, Berkeley, CA, Tech. Rep. M94/11, 1994.
[14] J.-H. R. Jiang and R. K. Brayton, "Functional dependency for verification reduction," in *Proc. Comput.-Aided Verification Conf.*, 2004, pp. 268–280.
[15] J. Krajicek, "Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic," *J. Symb. Log.*, vol. 62, no. 2, pp. 457–486, Jun. 1997.
[16] V. Kravets and P. Kudva, "Implicit enumeration of structural changes in circuit optimization," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2004, pp. 438–441.
[17] C.-C. Lee, J.-H. R. Jiang, C.-Y. Huang, and A. Mishchenko, "Scalable exploration of functional dependency by interpolation and incremental SAT solving," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2007, pp. 227–233.
[18] B. Lin and A. R. Newton, "Exact redundant state registers removal based on binary decision diagrams," in *Proc. Int. Conf. Very Large Scale Integr.*, 1991, pp. 277–286.
[19] M. Moskewicz, C. Madigan, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2001, pp. 530–535.
[20] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proc. Comput.-Aided Verification Conf.*, 2003, pp. 1–13.
[21] K. L. McMillan, *Symbolic Model Checking*. Norwell, MA: Kluwer, 1993.
[22] I.-H. Moon, J. Kukula, T. Shiple, and F. Somenzi, "Least fixpoint approximations for reachability analysis," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 1999, pp. 41–49.
[23] P. Pudlak, "Lower bounds for resolution and cutting plane proofs and monotone computations," *J. Symb. Log.*, vol. 62, no. 3, pp. 981–998, Sep. 1997.
[24] E. Sentovich, H. Toma, and G. Berry, "Latch optimization in circuits generated from high-level descriptions," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 1996, pp. 428–435.
[25] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Electron. Res. Lab, Univ. California, Berkeley, CA, Tech. Rep. UCB/ERL M92/41, May 1992.
[26] M. Syal and M. S. Hsiao, "VERISEC: Verifying equivalence of sequential circuits using SAT," in *Proc. High-Level Des. Validation Test Workshop*, 2005, pp. 52–59.
[27] C. A. J. van Eijk and J. A. G. Jess, "Exploiting functional dependencies in finite state machine verification," in *Proc. Eur. Des. Test Conf.*, 1996, pp. 9–14.
[28] J. Whittemore, J. Kim, and K. Sakallah, "SATIRE: A new incremental satisfiability engine," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2001, pp. 542–545.

Chen-Hsuan Lin received the B.S. and M.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2006 and 2008, respectively.

He is currently a Second Lieutenant with the Marine Corps, Taiwan. His research interests include logic synthesis and design verification.

Mr. Lin is a member of Phi Tau Phi.

Chun-Yao Wang (S'00–M'03) received the B.S. degree from the Department of Electronics Engineering, National Taipei University of Technology, Taipei, Taiwan, in 1994 and the Ph.D. degree from the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, in 2002.

He is currently an Associate Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu. His research interests include logic synthesis, design verification, and very large scale integration testing.

Yung-Chih Chen received the B.S. and M.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2003 and 2005, respectively, where he is currently working toward the Ph.D. degree in the Department of Computer Science.

His research interests include logic synthesis and design verification.