An Efficient Solution to the Storage Correspondence Problem For Large Sequential Circuits

Wanlin Cao Duncan M.H. Walker

Department of Computer Science
Texas A&M University
College Station, TX 77840
e-mail: {caowl,walker}@cs.tamu.edu

Rajarshi Mukherjee

Fujitsu Laboratories of America
595 Lawrence Expressway
Sunnyvale, CA 94087
e-mail rmukherj@fla.fujitsu.com

Abstract— Traditional state-traversal-based methods for verifying sequential circuits are computationally infeasible for circuits with a large number of memory elements. However, if the correspondence of the memory elements of the two circuits can be established, a difficult sequential verification problem can be transformed into an easier combinational verification problem. In this paper, we propose an approach that combines two *complementary simulation-based* methods for *fast* and *accurate* storage correspondence. Experiments on the large ISCAS89 benchmark circuits demonstrate the superiority.

I. Introduction

Traditional techniques to solve the problem of checking the equivalence of two sequential circuits involve state-space traversal of the two circuits or of their product machine using BDDs. If the state space is large, these methods could face the problem of memory explosion. However, if a correspondence can be established between the storage elements (flip-flops) of the two circuits, the two circuits can be verified using efficient combinational verification techniques [3].

Given two sequential circuits to be verified, it is often the case that a flip-flop correspondence exists, or is even known to the designers. But it is difficult to specify all the correspondences through a simple command or expression. So most commericial verification tools have an automatic flip-flop matching preprocessor that uses functional or ad-hoc methods based on signal names to match flip-flops. Ideally all the correspondences will be identified and reported by this preprocessor. If there remain some un-corresponded flip-flops, users can choose to specify all the correspondences manually or just provide some correspondences and then run the preprocessor again to find more correspondences. The functional flip-flop matching algorithms published so far are not robust. In fact none reports all the results for the big ISCAS 89 benchmark circuits. As for the ad-hoc methods, because tools that do design transformations such as clock tree insertion, or scan insertion, often do not preserve signal names, they may also fail in some cases. In such cases, designers have to specify the correspondence between flip-flops manually, resulting in inefficiency and errors. The goal of this paper is to develop a technique to find as many correspondences as possible within an expected running time limit. We will report results on all the big ISCAS 89 circuits. The main metrics of a flip-flop matching technique are the following: **(1)** *matching accuracy*, **(2)** *runtime*, and **(3)** *memory requirement*. Our experiments show that the proposed method outperforms both a sampling-based method and an ATPG-based method in all three respects.

The storage correspondence (flip-flop matching) problem can be defined as follows:

Definition I.1 *Given two sequential circuits, M_1 and M_2, with flip-flops $S = \{s_1, \ldots, s_n\}$ and $T = \{t_1, \ldots, t_m\}$ respectively, and having identical state encoding*[1]*, the automatic storage correspondence problem determines a set of groups of flip-flops, $G = \{g_1, \ldots, g_k\}$, such that each g_i contains one or more flip-flops from both circuits and the flip-flops in each group cannot be distinguished. Set G defines the correspondence between the flip-flops of the two circuits.*

The rest of the paper is organized as follows. Section II briefly outlines the existing techniques to solve the flip-flop correspondence problem. Section III presents the proposed algorithm. Experimental results are presented in Section IV. Section V presents the conclusions and future research directions.

II. Related Work

Several approaches for automatic storage correspondence exist in the literature. Broadly, these methods can be divided into the following three categories: **(1)** Simulation-based, **(2)** ATPG-based, and **(3)** BDD-based.

A random pattern simulation-based approach has been presented [4], where the two circuits are simulated from the initial states with randomly generated vectors applied to the primary inputs. Transitive fanin and fanout-based analysis is used to further enhance the capability of the method. This approach is deficient in matching accuracy because of its reliance on random pattern-based simulation.

A sequential ATPG-based state justification method for storage correspondence has been proposed [1]. This method operates on pairs of flip-flops in one circuit. Given two flip-flops

[1] Note that our definition allows the presence of redundant flip-flops in the circuits.

0-7803-6633-6/01/$10.00 ©2001 IEEE.

f_1 and f_2 to be distinguished, [1] models the problem as a test generation problem by inserting an XOR gate fed by f_1 and f_2. The two flip-flops cannot be distinguished if a s-a-0 fault at the output of this XOR gate is proved untestable by a sequential ATPG tool. If a test sequence is found, it is applied to both the circuits to divide the flip-flops into additional clusters. If there are redundant flip-flops in the circuits, this method can become expensive since it has to prove the corresponding faults untestable.

A fixed-point computation-based approach using BDDs has been proposed [5]. Here the flip-flops of the two circuits are initially assumed to be in the same cluster. Based on this assumption, the BDDs for the next-state functions are built. The existing clusters are further divided into new clusters based on the equality of the next-state function BDDs. This process continues until the clusters cannot be refined any further in two successive steps of computation (fixed-point). This method is prone to memory explosion due to its use of BDDs. Another method [8] has been proposed which uses sampling-based BDDs to compute equivalence groups of flip-flops. The use of *sampling* [8] reduces the possibility of memory explosion. However, the use of BDDs can still be time consuming and inefficient in terms of memory usage for large circuits.

Another BDD-based signature computation approach to identify flip-flop matching has been described in [7]. *Input signatures* and *output signatures* for the flip-flops are computed to establish flip-flop matching between two circuits.

Some other automatic storage correspondence techniques have been proposed in [6].

III. The Proposed Algorithm

A. Basis of the Proposed Algorithm

The proposed algorithm is based on the following two observations: (1) when a random pattern vector is applied on the primary inputs of two designs (the corresponding inputs will always be assigned the identical value), based on the responses (simulation results) of the next state lines, we can divide the flip-flops into two clusters, one includes all flip-flops with a logic value 0 on the next state lines and the other with all the flip-flops with logic value 1. If we apply yet another vector on the primary inputs and perform logic simulation, we can further divide each of the two clusters into two clusters. Ideally, after applying some random vectors, all the clusters will contain just two flip-flops, one from each design. Please note that, in the simulation process, all the flip-flops in a cluster will be assigned the same value on the current state lines. This observation tells us that, by applying a random vector on the primary inputs and the current state lines and observing the responses on the next state lines, we can divide flip-flops into more clusters and thus identify correspondences.

(2) For a cluster of flip-flops $s_1, s_2, ..., s_k, t_1, t_2, ..., t_k$, for the current state lines, let $s_1 = t_1 = 1$ and $s_2 = s_3 = ... = s_k = t_2 = t_3 = ... = t_k = 0$ and assign random values on the primary inputs. After logic simulation, if any of the corresponding primary output pairs has different responses, then we know s_1

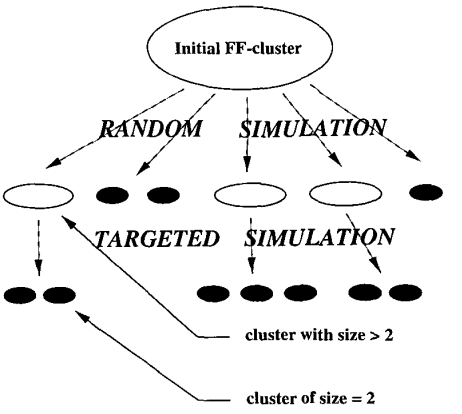

Fig. 1. Overview of Storage Correspondence Algorithm

```
// Routine assign_random_value() assigns random
// values to primary inputs and flip-flops.
// Corresponding primary inputs are assigned identical
// values. All flip-flops in a cluster are assigned the
// same value. Routine simulate() does two-valued
// parallel pattern simulation. Routine
// regroup_clusters() regroups current flip-flop
// clusters based on values at next-state functions.
// Routine all_matched() checks if all clusters
// contain two flip-flops, one from each circuit.
// Routine anomaly_detected() checks if any
// cluster has flip-flops only from one circuit.
// Routine limit_reached() checks if a predetermined
// limit on number of simulation cycles has been
// reached.

Rand_Sim(FF-clusters)
{
    assign_random_value ();
    simulate();
    regroup_clusters(FF-clusters);
    if (all_matched(FF-clusters)
        || anomaly_detected(FF-clusters)
        || limit_reached() )
      exit;
    else
      Random_sim(FF-clusters)
}
```

Fig. 2. *Random Pattern Simulation Algorithm*

```
// Routine continue_loop() checks if the algorithm will
// continue. It checks the following conditions:
// (1) if a cluster with more than two flip-flops remains
// (2) if a pre-determined number of patterns has been tried
// for inputs and flip-flops not in the cluster being processed.
// Routine assign_random_values() assigns random
// patterns to the primary inputs and flip-flops of
// the two circuits. Corresponding primary inputs are
// assigned identical values. All flip-flops in a
// cluster are assigned identical values. Routine
// get_cluster() gets the next cluster that has
// more than two flip-flops. Routine
// assign_targeted_values(C, i, s) assigns
// boolean value 1 to the flip-flop s_i in flip-flop
// cluster C and boolean value 0 to flip-flops s_l,
// l ≠ i. Routine simulate() does two-valued
// parallel simulation of the two circuits. Routine
// check_consistent() returns TRUE if
// the corresponding primary outputs and already
// matched flip-flops have identical responses, else
// returns FALSE. Routine no_match()
// marks a flip-flop as unmatched. Routine
// match () records a flip-flop match and divides
// the cluster further.

Targeted_Sim (unmatched-FF-clusters) {
    while (continue_loop()) {
        assign_random_values();
        C(s_1,...,s_k,t_1,...,t_k) = get_cluster();
        for (i = 1 to k) {
            count = 0; index = -1;
            assign_targeted_values(C, i, s);
            simulate(M_1);
            for (j = 1 to k) {
                assign_targeted_values(C, j, t);
                simulate(M_2);
                if (check_consistent())
                    count ++; index = j;
                }
            }
            if (count == 0)
                no_match (s_i); exit;
            if (count == 1)
                match(s_i, t_index);
        }
    }
}
```

Fig. 3. *Targeted Simulation Algorithm*

cannot match t_1. Otherwise we say s_1 and t_1 is a possible match. Of course, when we change the values on the primary inputs and perform simulation again, we may find a previously possible match becomes impossible. For s_i, if there is one and only one possible match t_j, then we know s_i and t_j is a match or there is no match for s_i and t_j. In this case, we report s_i and t_j as a match.

The differences between the two observations can be summarized as following: In observation 1, random vectors are applied on primary inputs and the current state lines. The responses on the next state lines are observed and used to distinguish flip-flops. With one pass (round) of logic simulation, we can potentially divide N clusters into $2N$ clusters. In observation 2, a specific vector is chosen to exclude some flip-flops from consideration. It targets one pair of flip-flops at a time. For a cluster that contains $2N$ flip-flops, in the optimal case, it needs $N+1$ rounds of simulation to find one match. If all the N^2 possible matches have been checked (by simulation) and no matches is found, a new random vector will be generated and applied on the prinmary inputs and current state lines of the flip-flops in other clusters and flip-flops in a cluster will be assigned the same value. Unlike observation 1, responses on the primary outputs or matched flip-flops are observed and compared. Whenever a difference on the corresponded primary outputs or flip-flops is found, the current simulation process will stop and next target pair of flip-flop will be simulated.

An algorithm based on these two observations has been developed. The algorithm consists of application of random pattern-based simulation followed by targeted simulation. Targeted simulation can be viewed as an intelligent *biased simulation*, where the biasing is done based on a target flip-flop that has to be distinguished from other flip-flops in a flip-flop cluster. This makes targeted simulation complementary in nature to random simulation and makes the combined approach powerful. Since the algorithm is based on simulation, it is extremely scalable and is easily applicable to very large sequential circuits with thousands of memory elements. Figure 1 shows the overview of our approach. The details of our approach are described below.

B. Random Pattern Simulation

Random pattern simulation is an effective method to quickly distinguish a large number of flip-flops. Figure 2 outlines our application of random pattern simulation.

Initially, all the flip-flops in the two circuits are grouped into one single cluster. Next, random patterns are applied to the primary inputs and the flip-flops. Corresponding primary inputs are assigned identical values. All the flip-flops in a cluster are assigned identical values since they have not yet been distinguished. The existing flip-flop clusters are further divided based on the values assigned to the next-state functions by simulation. This process continues until one of the following conditions is met: **(1)** a predetermined limit on the number of simulation cycles is reached, or **(2)** all flip-flop clusters contain exactly two flip-flops, one from each circuit (matched flip-flops),

or (3) if during the simulation, any cluster contains flip-flops only from one circuit. Flip-flops only from one circuit in a cluster indicate either a difference in the state encodings of the two circuits, or an error in one of the two circuits.

C. Targeted Simulation

Even though random pattern simulation can quickly distinguish a large number of flip-flops, due to its *non-targeted* nature, its distinguishing ability drops off after a large number of simulation cycles. Application of any further random patterns usually cannot distinguish any further flip-flops. This is when a state of *diminishing returns* is entered and a complementary technique should be applied to distinguish additional flip-flops. We have designed a targeted simulation-based approach, which is fast and has efficient memory usage, for this purpose. Given a set of flip-flop clusters that contain more than two flip-flops (flip-flops which could be further distinguished), we focus our attention to one cluster at a time. First, random patterns are assigned to the primary inputs (corresponding primary inputs are assigned identical values). Random patterns are also assigned to the flip-flops (all flip-flops in a cluster are assigned identical values). Next, each flip-flop cluster with more than two flip-flops is targeted as follows. Given a flip-flop cluster $C = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$, where each flip-flop s_i is from circuit M_1 and each flip-flop t_i is from circuit M_2, the algorithm shown in Figure 3 is applied to further distinguish them.

Given a flip-flop cluster $C = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$, **Targeted_Sim** has a complexity of $O(k^2)$ times circuit size (simulation time). While **Rand_Sim**() can potentially generate multiple new clusters with each round of simulation; **Targeted_Sim** can only generate one more new cluster per pass (with at least one round of simulation on M_1 and k rounds of simulation on M_2). To have a small k is important to the success of **Targeted_Sim**. This is exactly what **Rand_Sim**() can provide. Our experiments show that, after applying **Rand_Sim**() on the initial cluster that includes all the flip-flops, many small clusters will be generated. And **Targeted_Sim** can find new matches from the small clusters efficiently.

IV. Experimental Results

The proposed algorithm has been applied to the ISCAS 89 benchmark circuits. The experiments have been carried out on a SUN Ultra 30 workstation with 512MB memory. In the following experiments two identical copies of the ISCAS 89 benchmark circuits have been taken and flip-flops of one copy have been matched with the flip-flops of the second copy. Signal names and structural information, except PI and PO correspondences, have not been used during matching.

In Table I , we present results comparing random pattern simulation-based flip-flop matching (using only **Rand_Sim**()) with flip-flop matching using the proposed approach. A limit of 5000 vectors was imposed for random pattern simulation-based flip-flop matching. The proposed approach consists of application of random pattern-based simulation with a limit of 300 vectors followed by application of targeted simulation to the flip-flop clusters produced by random pattern simulation. Column 1 gives the name of the circuit. Columns 2,3, and 4 list the # primary inputs, # primary outputs, and # flip-flops respectively in the circuit. Column 5 lists the # vectors applied for random pattern simulation. Column 6 lists the average size of the flip-flop clusters at the end of the random pattern simulation. Column 7 lists the runtime in seconds for the random pattern simulation-based approach. Column 8 lists the number of random vectors applied in the proposed approach. Column 9 lists the # targeted vectors applied in the proposed approach. Column 10 lists the average size of the final flip-flop clusters after the application of the proposed approach. Column 11 lists the runtime of the proposed approach in seconds.

Most of the smaller circuits do not require targeted simulation. In Table I, we use "-" to represent cases where targeted simulation is not required, and hence the results obtained by the proposed approach are identical to those obtained by random pattern-based simulation. However, significant gains in runtime and matching accuracy[2] can be obtained by the proposed approach for the large circuits. We use 5000 vectors for the pure random simulation method such that the running time, compared with the proposed method, will not be too large. In fact we did make experiments with 50000 vectors for the random simulation method and there is only a small improvement in the results.

As mentioned earlier, targeted simulation is very efficient for small flip-flop clusters. In Table II we show that in the proposed approach the average cluster size after the application of 300 random vectors is indeed quite small. These small clusters are then fed to targeted simulation. Therefore, targeted simulation is extremely efficient.

Another interesting metric illustrating the efficiency of targeted simulation is shown in Figure 4. In the graph the x-axis denotes the number of vectors applied, and the y-axis denotes the average size of the flip-flop clusters after the application of each vector. We have plotted data obtained by applying random pattern simulation-based flip-flop matching and the proposed approach respectively to S9234. The graphs show that random pattern simulation is unable to distinguish a large number of flip-flops after the application of about 1500 vectors and the average cluster size saturates at about 2.23. On the other hand, application of about 700 targeted vectors (applied after first applying 300 random vectors) rapidly distiguishes a large number of flip-flops and reduces the average cluster size to about 2.18. This graph shows that the proposed approach achieves improved flip-flop matching speed and accuracy by properly combining the complementary capabilities of random pattern-based simulation and targeted simulation.

In Table III we compare the proposed method with sampling-based flip-flop matching [8] and ATPG-based flip-flop matching [1] for the large circuits in the ISCAS 89 benchmark suite. The results show that the proposed approach has

[2]The closer the average final flip-flop cluster size is to 2, the higher the matching accuracy.

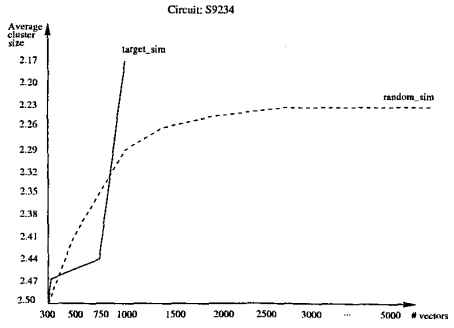

Fig. 4. Comparison of Efficiency of Random and Targeted Simulation

circuit	# clusters of size > 2	ave. size
S5378	1	4
S9234	22	6
S13207	61	7.54
S15850	25	6.56
S35932	32	18
S38417	78	10.72
S38584	34	8.71

TABLE II
CLUSTERS FED TO TARGETED SIMULATION

better matching accuracy in a majority of circuits and has faster runtimes. In addition, since the memory requirement of simulation is proportional to the size of the circuit, the proposed approach has lower memory requirement than the sampling-based approach, which is prone to memory explosion. In Table III, "-" marks the cases where results are not available, and "abort" marks cases where the BDD sizes exceed the given memory limit.

V. CONCLUSIONS

In this paper we have presented an efficient, scalable and accurate method for storage correspondence on large sequential circuits. The proposed approach combines two complementary methods, namely, random pattern-based simulation and targeted simulation to achieve improved performance. Experimental results on a large number of ISCAS 89 benchmark circuits show that this method outperforms the sampling-based [8] and ATPG-based [1] approaches for storage correspondence. The proposed approach is easily applicable to very large circuits with thousands of flip-flops unlike the BDD-based and ATPG-based approaches proposed in the literature. In addition, application of targeted simulation makes the approach much more efficient and accurate than random pattern-based simulation alone.

As future research, we would like to address situations where application of the proposed approach becomes computationally expensive or ineffective. In such cases, we can use ATPG to replace target simulation in the following way. For a cluster of flip-flops, s_1, s_2, ..., s_k, t_1, t_2, ..., t_k, we use ATPG to check if s_i XOR t_j is s-a-0 testable. If yes, we know s_i and t_j cannot be a match. If s_1 cannot be matched with t_2, ..., t_k, then we say s_1, t_1 is a match. The ATPG search will be used in a limited manner so that it will not spend much time on cases where s_i XOR t_j is s-a-0 untestable.

REFERENCES

[1] V. Boppana, R. Mukherjee and M. Fujita, "Storage Correspondence For Large Sequential Circuits", Proc. International Workshop on Logic Synthesis, Lake Tahoe, CA, 1998.

[2] J. R. Burch and V. Singhal, "Robust Latch Mapping for Combinational Equivalence Checking", Proc. ICCAD 1998, San Jose, CA. pp.563-569.

[3] J. R. Burch and V. Singhal, "Tight Integration of Combinational Verification Methods", Proc. ICCAD 1998, San Jose, CA., pp. 570-576.

[4] H. Cho and C. Pixley, "Apparatus and Method for Deriving Correspondence Between Storage Elements of a First Circuit Model and Storage Elements of a Second Circuit Model" U.S. Patent, No. 5,638,381, June 1997.

[5] C. van Eijk,"Formal Methods for the Verification of Digital Circuits", Ph.D. Thesis, Eindhoven University of Technology, 1997.

[6] T. Filkorn, "Symbolische Methoden fur die Verifikation endlicher Zustandssysteme", Dissertation Institut fur Informatik der Technishen Universitat Munchen, 1992.

[7] J. Mohnke, P. Molitor and S. Malik, "Establishing Latch Correspondence for Sequential Circuits Using Distinguishing Signatures", INTEGRATION, the VLSI journal, 27:33-46, January 1999.

[8] R. Mukherjee, V. Boppana, J. Jain and M. Fujita, "Efficient Verification of Sequential Circuits with Same State Encoding", Technical Report, Fujitsu Laboratories of America, 1999.

circuit	# in	# out	# FF	random simulation			**proposed approach**			
				# vec	ave. size	time	# rand. vec.	# targeted vec.	ave. size	time
S208	12	3	8	9	2	0.03	-	-	-	-
S298	3	6	14	4	2	0.02	-	-	-	-
S344	9	11	15	2	2	0.01	-	-	-	-
S349	9	11	15	2	2	0.01	-	-	-	-
S382	3	6	21	14	2	0.05	-	-	-	-
S386	7	7	6	14	2	0.05	-	-	-	-
S400	3	6	21	11	2	0.05	-	-	-	-
S510	19	7	6	2	2	0.02	-	-	-	-
S526	3	6	21	28	2	0.10	-	-	-	-
S641	35	23	19	5000	2.235	15.55	300	4	2	1.10
S713	35	23	19	5000	2.235	17.61	300	4	2	1.47
S820	18	19	5	3	2	0.05	-	-	-	-
S832	18	19	5	4	2	0.05	-	-	-	-
S1423	17	5	74	5000	2.027	52.32	300	2	2	3.96
S1488	8	19	6	3	2	0.04	-	-	-	-
S1494	8	19	6	3	2	0.04	-	-	-	-
S5378	35	49	164	5000	2.012	130.70	300	160	2.012	8.59
S9234	36	39	211	5000	2.245	215.78	300	688	2.186	36.01
S13207	62	152	633	5000	2.584	387.78	300	3040	2.213	214.71
S15850	77	150	533	5000	2.220	432.56	300	540	2.132	41.12
S35932	35	320	1770	5000	2.348	1125.30	300	2980	2.214	548.21
S38417	28	106	1636	5000	2.282	1206.40	300	1718	2.260	389.62
S38584	38	304	1426	5000	2.174	1415.00	300	1214	2.006	274.62

TABLE I
RANDOM SIMULATION VS. PROPOSED APPROACH

circuit	sampling [8]		ATPG [1]		**proposed approach**	
	ave. size	*time*	*ave. size*	*time*	*ave. size*	*time*
S820	2	0.02	2	0.19	2	0.05
S832	2	0.02	2	0.20	2	0.05
S1423	2	1.12	2	353.30	2	3.96
S1488	2	0.01	2	0.54	2	0.03
S1494	2	1.00	2	0.60	2	0.03
S5378	2.012	35.46	3.93	2803.37	2.012	8.59
S9234	2.245	66.62	-	-	2.186	36.01
S13207	2.380	334.80	-	-	2.213	214.71
S15850	2.086	477.19	-	-	2.132	41.12
S35932	abort	abort	-	-	2.214	548.21
S38417	abort	abort	-	-	2.260	389.62
S38584	abort	abort	-	-	2.006	274.62

TABLE III
COMPARISON WITH SAMPLING AND ATPG-BASED METHODS