Sequential Equivalence Checking Using Cuts

Wei Huang
State Key Laboratory of
ASIC&System(Fudan University)
Shanghai 200433,China
dennyhuang@fudan.edu.cn

PuShan Tang
State Key Laboratory of
ASIC&System(Fudan University)
Shanghai 200433,China
pstang@fudan.edu.cn

Min Ding
State Key Laboratory of
ASIC&System(Fudan University)
Shanghai 200433,China
mding1977@163.com

Abstract This paper presents an algorithm which is an improvement of Van Eijk's Algorithm[5] by incorporating a cutpoints technique[8]. Combinational verification often uses the technique to convert large scale circuits to several small ones, which will be verified separately. Reasonable cuts can bring less time consuming to combinational verification. We embed the technique into sequential equivalence checking. Experimental results show that the proposed method can achieve about 2x speedup over the original one.

I. Introduction

Sequential equivalence checking is an open problem in electronics design automation. Impressive progress has been made in this domain by symbolic approaches[2], which are based on BDDs[1] .Unfortunately, BDDs-based method often suffers from memory explosion. Various approaches have been proposed to avoid memory explosion by exploiting structural similarities[3][4][5]. These approaches have had great success in combinational equivalence checking. For sequential equivalence checking there have been only a few approaches using the technique of exploiting structural similarities.

In [5], Van Eijk has proposed an algorithm on sequential equivalence checking. The main idea of the algorithm is to find the nodes which have same value(or opposite value) in all reachable states of the two circuits. This information can sometimes directly speculate that the two circuits under verification are sequentially equivalent.

The information is called an equivalence relation over all the nodes in all reachable states of the circuits. First ,The algorithm computes an equivalence relation T_0 in initial state s_0 .Then it keeps improving the equivalence relation T_i ($i \geq 0$) and forming new equivalence relation T_{i+1} until they are equal. It can be proved that the final equivalence relation T_{final} holds in all reachable states.

Most of the running time of Van Eijk's Algorithm is consumed on the improvements of the equivalence relations. In order to compute an improved equivalence relation T_{i+1} from a relation T_i ,the Algorithm first assumes the equivalence relation T_i holding in current time instance, then checks the equivalences described by T_i (i.e. compares the functions of the nodes in the same equivalent class) in the next time instance, finally forms T_{i+1} .When comparing the functions in the same equivalent class, the algorithm does not consider the association between different comparisons, but does them independently. Our algorithm sufficiently uses the association between different comparisons, reasonably arranges every comparison of the δ functions, embeds the cutpoints technique into Van Eijk's Algorithm. Experimental results show that our improved algorithm can achieve 2x speedup over the original one.

The organization of this paper is as follows. Section II reviews Van Eijk's Algorithm on sequential equivalence checking. Section III introduces our improved algorithm. We report the experimental results in section IV and conclude the paper in section VI.

II. VAN EIJK'S Algorithm on Sequential Equivalence Checking

The model of a sequential circuit is **finite state machine**(FSM) with a initial state s_0 . It is defined as a 6-tuple $M=(X,Y,S,s_0,\Delta,\Lambda)$,where X is the input alphabet,Y is the output alphabet, S is the set of states, $s_0 \in S$ is an initial state, $\Delta: X \times S \rightarrow S$ is the next-state function, and $\Lambda: X \times S \rightarrow Y$ is the output function.

Product machine: Given two FSMs $M_A=(X,Y,S,s_{A.0},\Delta_A,\Lambda_A)$ and $M_B=(X,Y,S,s_{B.0},\Delta_B,\Lambda_B)$, the product machine formed by M_A and M_B is $M=(X,Y,S,s_0,\Delta,\Lambda)$, where $S=S_A \times S_B$, $s_0=s_{A.0} \times s_{B.0}$, $\Delta(S_A,S_B,X)=(\Delta_A(S_A,X),\Delta_B(S_B,X))$, and $\Lambda(S_A,S_B,X)=(\Lambda_A(S_A,X) \equiv \Lambda_B(S_B,X))$.

We realize that the two FSMs are equivalent iff every output of their product machine keeps the Boolean value 1 in all reachable states.

As discussed above, Van Eijk's Algorithm try to compute an equivalence relation T_{final} in all reachable states. First ,it computes between the nodes an equivalence relation T_0 in initial state s_0 . Then it improves the equivalence relation by assuming that the equivalences computed thus far hold in current time instance and deriving the subset of these equivalences that must hold in the next time instance. It iterates the improvement until the equivalence relation can not be improved any longer. The final equivalence relation is satisfied in the initial state, and preserves any state transition. Therefore, it must hold in all reachable states.

Before we give a detailed description of the algorithm , let's introduce some definitions.

State Compatibility condition. Given an equivalence relation T on the nodes F of a product machine. State compatibility condition is a function:

$$Q_T=(\forall x \in X: \prod_{f_m,f_n \in F \wedge f_m T f_n} f_m(s,x)=f_n(s,x)) \quad ...(1)$$

which defines whether a state's conforms to T ,i.e., whether all nodes in the same equivalence class of T indeed have the same Boolean value in the state s.

Time Frame Model of a product machine: as shown in Fig 1, it contains two consecutive time frames, which are called current time frame and next time frame respectively. Under the model, two functions are associated with each node v: current-state function $f_v : S \times X \to B$ which expresses the Boolean value of v in the current time frame as a function of the current state and current input vector; and next-state function $\delta_v : S \times X \times X \to B$ which expresses the Boolean value of v in the next time frame as a function of the current state, the current input and the next input vector. Note that:

$$\delta_v(S_i, X_i, X_{i+1}) = f_v(S_{i\_next}, X_{i+1}) = f_v(\Delta(S_i, X_i), X_{i+1}) \quad \text{.....(2)}$$

Equation (2) shows that the next-state function δ_v implicitly include state transition relation.

Algorithm. It computes an equivalence relation T_0 in the initial state s_0. For node m and n, $f_m T_0 f_n$ according to the following formula:

$$f_m T_0 f_n \Leftrightarrow \forall x \in X : f_m(s_0, x) = f_n(s_0, x) \quad \text{.....(3)}$$

Starting with T_0, a sequence of equivalence relation is computed by applying the following formula:

$$f_m T_{i+1} f_n \Leftrightarrow f_m T_i f_n \wedge (\forall s \in S, x_i, x_{i+1} \in X : Q_{T_i} \Rightarrow \delta_m(s, x_i, x_{i+1}) = \delta_n(s, x_i, x_{i+1})) \quad \text{.....(4)}$$

Based on the time frame model of the product machine, equation (4) describes the computation of $T_{i+1}(i \geq 0)$. For node m and n, $f_m T_{i+1} f_n$ iff the following two conditions are satisfied:

a) $f_m T_i f_n$

b) assuming that T_i is valid in the current time frame of the product machine, the next-state functions of m and n have the same Boolean value.

The iteration of computing $T_{i+1}(i \geq 0)$ is complete if it is equal to T_i ,which means T_i can not be improved any longer and a fixed point is reached.

It can be proved[5] that the final equivalence relation T_{final} computed according to the foregoing steps holds in all reachable states. The algorithm is a kind of approximate methods and suffers from false negatives.

III. The Improved Algorithm

Disadvantages of Van Eijk's Algorithm. We use the example of Figure 2 to discuss the improvement of equivalence relation. Figure 2(a) and 2(b) depict two structurally different but functionally equivalent circuits with initial state $V_1 = 0$ and $V_6 = 0$. Figure 2(c) depicts the time frame model of their product machine.

Suppose that for a certain time instance i, there is an equivalence relation:

$T_i = \{\{f_1, f_6\}, \{f_3, f_8\}, \{f_4, f_9\}, \{f_5, f_{10}\}\}$.

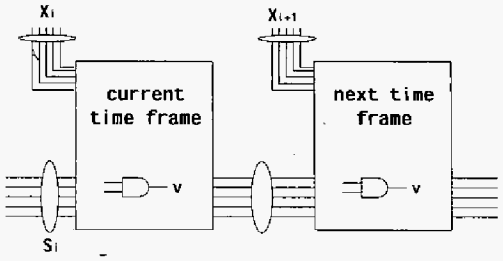

Fig.1. Time frame model of the product machine.

SAT-Solver is chosen as the engine of the algorithm .The state compatibility condition of T_i is:

$Q_{T_i} = (f_1 = f_6) \bullet (f_3 = f_8) \bullet (f_4 = f_9) \bullet (f_5 = f_{10})$.

In order to get T_{i+1} , δ_1 and δ_6 , and δ_8 , δ_4 and δ_9 , δ_5 and δ_{10} need to be compared. Some simple logic relations and their corresponding CNF(conjunctive normal form) formulas of SAT-Solver are shown in the following table:

Logic Relation	Corresponding CNF
c=AND(a,b)	$(\bar{c}+a)(\bar{b}+a)(c+\bar{a}+\bar{b})$
c=OR(a,b)	$(c+\bar{a})(c+\bar{b})(\bar{c}+a+b)$
b=NOT(a)	$(a+b)(\bar{a}+\bar{b})$
c=NAND(a,b)	$(c+a)(c+b)(\bar{c}+\bar{a}+\bar{b})$
c=NOR(a,b)	$(\bar{c}+\bar{a})(\bar{c}+\bar{b})(c+a+b)$
b=a	$(\bar{b}+a)(b+\bar{a})$

For the comparison of δ_1 and δ_6 ,the CNF representation of associated problem contains the following components:

a) CNF formula for Q_{T_i} ,which contains $4 \times 2 = 8$ clauses.

b) CNF formula for the fan-in nodes of δ_1 , δ_6 ; f_1 , f_6 , f_3 , f_8 , f_4 , f_9 , f_5 and f_{10} . They are the nodes of X_i , f_1 , f_2 , f_3 , f_4 , f_5 , f_6 , f_7 , f_8 , f_9 , f_{10} , δ_1 and δ_6 . According to their logic relations, there are $0+0+3+3+3+3+0+2+3+3+3+2+2=27$ clauses .

c) CNF formula for $\delta_1 = NOT(\delta_6)$, under which condition a *SATISFIABLE* assignment means there exists an input vector of V_1 , V_6 , X_i and X_{i+1} that can make δ_1 and δ_6 not equal. Otherwise, they are equal. The formula contains 2 clauses.

Therefore, the total number of SAT clauses needed to compare δ_1 and δ_6 is $8+27+2=37$.

Similarly, the total number of SAT clauses needed to compare other δ functions can be achieved, which is outlined in the second column of TABLE I.

From the foregoing analysis, we inspect that during the computation of X_{i+1} , Van Eijk's Algorithm does not consider the association between different comparisons of δ functions, i.e., every comparison is performed

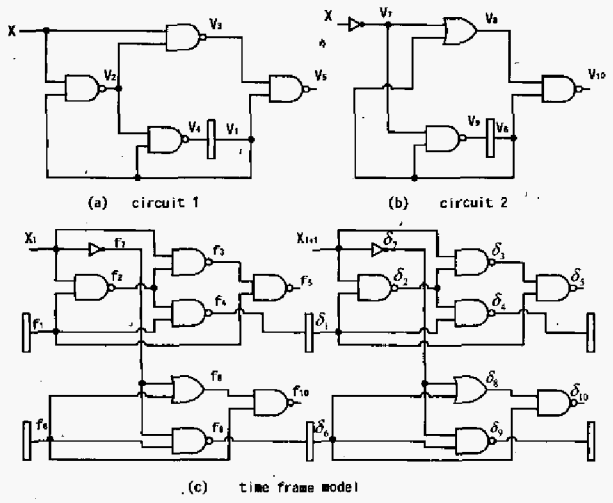

Fig.2. Two circuits under verification and their time frame model

independently.In fact, the association does exist. Further looking into the example of Figure 2, we find out that the clauses needed to compare δ_3 and δ_8 include the ones needed to do δ_1 and δ_6. Therefore, if the comparison of δ_1 and δ_6 is fulfilled, the information of the result can be shared by the comparison of δ_3 and δ_8, which will achieve a great speedup. Similarly, the comparison of δ_4 and δ_9 can share the information of the comparison of δ_1 and δ_6 too. And so can δ_5 and δ_{10}.

In the following, the advantage of enjoying the association of different comparisons will be discussed. Suppose δ_1 and δ_6 have been proved equivalent, cut frontiers can be made through the nodes of δ_1 and δ_6. Then, for the comparison of δ_3 and δ_8, the CNF representation of associated problem contains the following components:

a) CNF formula for $\delta_1 = \delta_6$,which contains 2 clauses.
b) CNF formula for the corresponding logic gates of δ_1, δ_2, δ_3, δ_6, δ_7 and δ_8 ,which contain $0+3+3+0+2+3=11$ clauses.
c) CNF formula for $\delta_3 = NOT(\delta_8)$,which contains 2 clauses.

Therefore, the total number of clauses needed to compare δ_3 and δ_8 is $2+11+2=15$ after cut frontiers are made. Similarly, the comparison of δ_4 and δ_9 needs $2+11+2=15$ clauses.

Furthermore, the comparison of δ_5 and δ_{10} needs $4+3+2=9$ clauses when cut frontiers are made through the nodes of δ_3, δ_4, δ_8 and δ_9.

The number of clauses needed to compare different δ functions before and after cut frontiers is outlined in TABLE I.

From TABLE I, we inspect that if the cutpoints technique[8] is adopted, the associated problem for the comparison of δ functions can be made easier to solve.

Cutpoints technique applied in Van Eijk's Algorithm.

The pseudo-code for the improved algorithm on computing T_{i+1} is as follows:

```
CalculateTnextUsingCUTs( T_i ){
    while(np = LowestNodePair( T_i )){
        cutFrontiers = NULL;
        while(result==SATISFIABLE
              &&!ExceedGivenLimit(cutFrontiers){
              cutFrontiers = GenerateCutFrontiers
              (np,cutFrontiers);
               result = CompareBasedOnCuts
                        (cutFrontiers, np);}
         if(result==SATISFIABLE){
             CompareNodePairExactly(np);
        }  }
    T_{i+1} = GenerateNewEquRelation();
    Return T_{i+1} ; }
```

The improved algorithm proposed here is based on cutpoints technique. Every comparison of δ functions is based on the comparisons of their "fan-in" δ functions if they have. Therefore, the comparisons of δ functions should be firstly performed on those nodes which are very close to the primary input nodes, and the δ functions of the farther ones are compared later. The procedure LowestNodePair() is to choose a pair of nodes which are closest to the primary input nodes and have not been compared.

When comparing the δ functions of a given pair of nodes, we perform backtrack towards the primary input nodes and form cut frontiers at those *already-proved* equivalent nodes. Based on the cuts, a complicated comparison can be transferred into a simple one. The procedure CompareBasedOnCuts() is to compare the δ functions of the pair of nodes based on a given cuts. It has two return values: *SATISFIALBE* or *UNSATISFIABLE*. When it is *UNSATISFIABLE*, the two compared nodes are equivalent. When it is *SATISFIALBE*, due to false negative[8], the situation is complicated: if the cut frontiers have totally reached the primary input nodes, the two nodes are real unequal, otherwise, if they have not, the cut frontiers need to be backtracked. In addition, the backtrack should be limited by a threshold because too many backtracks may be unnecessary. The procedure GenerateCutFrontiers() is to generate new cut frontiers from given ones. The procedure

Table I. Number of clauses before and after cuts

δ function	Clauses (Van Eijk)	Clauses(using cuts)
δ_1 and δ_6	$8+27+2=37$	$8+27+2=37$
δ_3 and δ_8	$8+38+2=48$	$2+11+2=15$
δ_4 and δ_9	$8+38+2=48$	$2+11+2=15$
δ_5 and δ_{10}	$8+44+2=54$	$4+3+2=9$

ExceedGivenLimit() is to check whether the newly-generated cut frontiers have exceeded a given threshold or not. If it returns false, the comparison based on the cut frontiers will be performed, otherwise, it goes to another branch CompareNodePairExactly(), which is a procedure that can perform the comparison of two δ functions austerely abiding by Equation (4) and tell us whether they are equivalent or not. There doesn't exist a best threshold for all benchmarks and we can only set up a relatively good one according to experimental results.

Moreover, as mentioned in Van Eijk's Algorithm, random simulation[6] can be used as a preprocess to partition the nodes of the product machine into a set of equivalent classes and form an equivalence relation T_{random} before T_0 is calculated, which can bring less time consuming.

IV. Experimental Results

Using the SIS [7] environment, we implemented the improved algorithm. The experiments were conducted on a 450 MHz Sun Blade1000 workstation with 2 GBytes memory. Benchmarks from ISCAS'89 are selected to test the algorithm. The original benchmark descriptions are verified against the synthesized versions of them, which are optimized by fx of SIS.

As introduced before, we set up a threshold to control the backtrack of cut frontiers. Large numbers of experiments demonstrate that the average running time of the whole algorithm is least when we set up the maximum degree of backtrack as 5 for each comparison.

Table II shows the experimental results. The first column shows the name of the benchmark. The second column shows number of iterations of computing a final equivalence relation . The following column shows the running time of random simulation. The fourth and fifth columns list the running time of fixed point iteration without and with cutpoints technique. The last column shows the multiple of improvement on running time of two different methods of fixed point iteration.

The experimental results clearly show that the improved algorithm we propose is able to achieve about 2x speedup over the original one. Specially, for the benchmark of S1488 and S991, the proposed algorithm can achieve 4.45 and 5.24 speedup respectively.

Table II. Experimental results on ISCAS'89 benchmarks

Circuit	# its	RS (s)	TWOC(s)	TWC (s)	Multiple
S208	5	0.05	0.31	0.24	1.29
S298	2	0.16	0.16	0.13	1.23
S344	2	0.18	0.35	0.17	2.06
S349	2	0.20	0.35	0.18	1.94
S382	4	0.33	0.65	0.50	1.30
S400	11	0.39	1.95	1.73	1.13
S420	23	0.29	6.49	4.09	1.59
S444	3	0.35	0.57	0.35	1.63
S499	1	0.09	0.29	0.15	1.93
S510	1	0.19	0.44	0.14	3.14
S526	8	0.47	2.07	1.71	1.21
S641	2	0.30	0.58	0.45	1.29
S838	64	1.66	95.74	65.10	1.47
S967	4	0.05	5.58	2.97	1.88
S991	2	0.03	5.45	1.04	5.24
S1196	2	0.86	5.32	2.99	1.78
S1238	2	0.98	6.19	3.47	1.78
S1269	6	0.07	35.21	14.73	2.39
S1423	13	6.58	98.71	38.53	2.56
S1488	1	0.43	5.70	1.28	4.45
S1494	5	0.02	29.48	11.56	2.55
S1512	15	0.17	41.92	29.09	1.44
S3271	2	0.72	42.32	16.79	2.52
S3330	3	0.32	71.71	64.81	1.11
S4863	2	0.72	90.19	33.42	2.70
S6669	12	3.71	590.07	214.96	2.75

VI. Conclusion

We have improved Van Eijk's Algorithm presented in [5] by incorporating the cutpoints technique. In order to deal with false negative accompanied the technique, a threshold has been set up after numerous experiments to control the backtrack of cut frontiers. Experimental results show that the proposed algorithm can achieve about 2x speedup over the original one.

Acknowledgements

This research is supported by NSFC Grant No. 90207002, and National High Technology Research and Development 863 Plan under contract 2002AAIZ1460, and founded partly by Synopsis InC.

References

[1]. Hu,A.J., "Formal Hardware Verification with BDDs: An Introduction" Communications, Computers and Signal Processing,1997.'10 Years PACRIM 1987-1997-Networking the Pacific Rim'.1997 IEEE Pacific Rim Conference,Volume:2, Pages: 677-682, Aug.20-22,1997.

[2]. H.J.Touati, H.Sarvoj ,B.Lin, R.K.Brayton and A.Sangiovanni-Vincentelli, "Implicit State Enumeration of Finite State Machines Using BDD's,"Proc.Int'l Conf. On Computer-Aided Design, pp.130-133, Nov.1990.

[3]. S.-Y.Huang, K.-T.Cheng, and K.-C.Chen, "AQUILA: An Equivalence Verifier for Large Sequential Circuits,"Proc. of Asia and South Pacific Design Automation Conf,pp.134-139, April 1997

[4]. D.Brand, "Verification of large synthesized designs" in Proc. Int. Conf. Computer-Aided Design,pp.534-537,1993.

[5]. C.A.J.Van Eijk "Sequential Equivalence Checking Based on Structural Similarities" IEEE.Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol.19, no.7, pp.814-819, July 2000.

[6]. F.Krohm,A.Kuehlmann. and A.Mets. "The Use Of Random Simulation in Formal Verification" Proc. Of Design Automation Conference,pp.218-223,Jun.1994.

[7]. Ellen M. Sentovich, Kanwar Jit Singh, Luciano Lavagno, Chao Moon, Rajeev Murgai, Alexander Saldanha, Hamid Savoj, Paul R.Stephan, Robert K.Rrayton, Alberto Sangiovanni-Vincentelli, "SIS:A System for Sequential Circuit Synthesis", Technical report, University of California at Berkeley Memorandum No.UCB/ERL M92/41 ,1992.

[8]. A. Kuehlmann and F. Krohm, "Equivalence Checking Using Cuts and Heaps ", In Design Automation Conference,pp.263-268,Jun1997.

[9]. Silva,J.O.M.;Sakallah,K.A." Robust Search Algorithm for Test Pattern Generation" Fault-Tolerant Computing,1997.FTCS-27.Digest of Papers,Twenty-Seventh Annual International Symposium, pp.152-161, June 24-27, 1997.