On Equivalence Checking and Logic Synthesis of Circuits with a Common Specification

Cadence Berkeley Labs
1995 University Ave.,Suite 460, Berkeley, California,94704
phone: (510)-647-2825, fax: (510)-486-0205

CDNL-TR-2004-1220
December 2004

Eugene Goldberg (Cadence Berkeley Labs), egold@cadence.com

Abstract. *In this report we develop a theory of equivalence checking and logic synthesis of circuits with a common specification (CS). We show that two combinational circuits N_1,N_2 have a CS iff they can be partitioned into subcircuits that are connected "in the same way" and are* ***toggle equivalent****. This fact allows one to represent a specification of a circuit implicitly as a partitioning into subcircuits. We give an efficient procedure for checking if circuit N_1, N_2 have the same predefined specification. As a "by-product", this procedure checks N_1 and N_2 for functional equivalence. We show how, given a circuit N_1 with a predefined specification, one can efficiently build a circuit N_2 satisfying the same specification. We give experimental evidence that equivalence checking of N_1, N_2 is hard if their CS is unknown. We also show experimentally that one can eliminate logic redundancy of circuit N_1 by building a circuit N_2 that is toggle equivalent to N_1.*

1. Introduction

In this report we continue developing the theory of equivalence checking and logic synthesis of circuits with a common specification (CS) started in [5][6][7]. A CS S of combinational circuits N_1 and N_2 is just a circuit of multi-valued gates (further referred to as ***blocks***) such that N_1 and N_2 are different implementations of S. Figure 1 gives an example. Circuits N_1 and N_2 have a 3-block CS shown on the left. Subcircuits N_1^i, N_2^i are different implementations of the multi-valued block G_i of S. Circuit N_m^i (m=1,2) implements a multi-output Boolean function whose truth table is obtained from that of G_i by replacing values of multi-valued variables with their binary codes. So the difference between N_1^i and N_2^i is in the choice of binary encodings for the variables of S. The size of the largest subcircuit N_m^i is called the ***granularity*** of specification S of N_m.

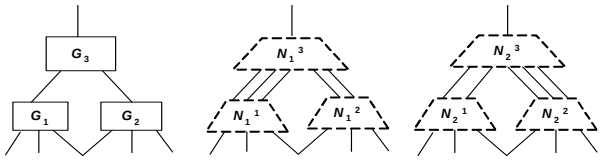

Figure 1. Circuits N_1 and N_2 with a common specification of three blocks

There are at least four reasons why a theory of circuits with a CS is of practical importance: a) large circuits are usually composed of a set of meaningful subcircuits; b) the space of implementations the same specification is very "rich" and so this space most likely contains implementations that are much better than the original one; c) there is an efficient procedure for checking if N_1 and N_2 implement the same predefined specification and are functionally equivalent; d) there is an efficient procedure to build a circuit satisfying a predefined specification.

An example of a large combinational circuit having a high-level structure is a multiplier that is usually specified as a network of smaller subcircuits, adders. A sequential circuit always has a natural partitioning into combinational subcircuits implementing next-state functions. (Although the theory we develop in this report targets synthesis and verification of large combinational circuits with a high-level structure, after some modification, it can be also applied to synthesis and verification of sequential circuits.).

Implementations of the same specification S are different only in the choice of encodings. A k-output subcircuit can be considered as an implementation of a 2^k-valued function. The

number of different k-bit encodings of a 2^k-valued variable is $(2^k)!$. This number is very large even for very small values of k. So even for a specification S of very small granularity, the space of implementations is huge.

One of the problems one has to face when building a circuit N_2 that is another implementation of a specification S of N_1 is the increasing complexity of equivalence checking (EC). Since N_1 and N_2 may not have any functionally equivalent internal points, their equivalence cannot be, in general, efficiently checked by the existing algorithms. (The most efficient equivalence checkers heavily rely on existence of functionally equivalent internal points.) Fortunately, in [5][6] it was shown that if a CS S of N_1 and N_2 is known, there is an efficient procedure for EC of N_1 and N_2. The most advanced version of this procedure was given in [7]. The advantage of the procedure of [7] is that one does not need to know either the functionality of multi-valued blocks or binary encodings. A CS S of N_1 and N_2 is represented as partitioning of N_1 and N_2 into subcircuits. This procedure has exponential complexity in the granularity of the CS S of N_1 and N_2 and linear complexity in the number of subcircuits (i.e. blocks of specifications). So if one considers only EC of circuits N_1, N_2 with a CS of granularity bounded by a constant, the complexity of the procedure of [7] is ***linear*** in circuit size.

The flaw of the procedure of [7] is in its relying on the assumption that the specifications S_1 of N_1 and S_2 of N_2 represented by the corresponding partitions are identical i.e. S_1=S_2=S. In this report, we show that circuits N_1 and N_2 have a CS of a specified topology T iff they can be partitioned into subcircuits N_1^1,.., N_1^k and N_2^1,.., N_2^k that are connected as described by T and if N_1^j and N_2^j are toggle equivalent j=1,..,k. We also modify the procedure of [7] so that now it not only checks if N_1 and N_2 are functionally equivalent but also tests if the specification of N_1 and that of N_2 are identical. The modified procedure has the same complexity as that of [7].

As it was mentioned above the space of implementations of a specification S is ***huge*** even if S has small granularity. So, given an implementation N_1 of specification S that does not satisfy necessary requirements, there is a good chance to find an implementation N_2 of S that will meet our requirements. The notion of toggle equivalence is key in building such a circuit N_2. In this report, we show that if S is specified as a partitioning of N_1 into subcircuits N_1^1,.., N_1^k, to build N_2 it is sufficient to replace each subcircuit N_1^i with its toggle equivalent counterpart N_2^i.

The importance of the notion of toggle equivalence it that it allows one to reencode multi-valued variables ***implicitly*** without any knowledge of the functionality of multi-valued blocks and/or binary encodings of multi-valued variables. One reason why re-encoding of multi-valued variables implicitly is a good idea is that the domain of a multi-valued variable may be too large to be handled explicitly. (For example, a 20-output subcircuit of N_1 can be considered as an implementation of a multi-valued block whose output variable takes up to 2^{20} values.) Another important reason for using implicit reencoding is that it is hard to "guess" a good encoding at the level of multi-valued blocks. Besides, when re-encoding a multi-valued variable it is reasonable to try to improve the current encoding rather than start building a new encoding from scratch. This can be done if one finds a way to modify a circuit implementing a multi-valued function in such a way that the modified circuit is still an implementation of this function.

In this report, we show that if two multi-output subcircuits N and N' are toggle equivalent, they are implementations of the same multi-valued function. This means that if N is an implementation of a multi-valued function g and we want to re-encode the output variable of g it sufficient to synthesize a circuit N' that is toggle equivalent to N. If circuit N has k outputs it means that by building the circuit N' we reencode a variable taking up to 2^k values. Besides, if, when building the circuit N' one optimizes N (rather than builds N' from scratch), the new encoding is obtained by "improvement" of the previous encoding rather than by building a completely new one.

The report is structured as follows. Section 2 introduces the notion of toggle equivalence of Boolean functions. In Section 3 we show that circuits N_1 and N_2 have a CS iff they can be partitioned into toggle equivalent subcircuits that are connected "in the same way" in N_1 and N_2. In Section 4 we give a procedure for checking if a CS of N_1, N_2 is correct. In Section 5 we explain why EC of circuits with unknown CS is hard. In Section 6 we discuss logic synthesis of circuits preserving a predefined specification. In Section 7 we give experimental evidence that EC of circuits with unknown CS is hard. Besides, we show that one indeed can use toggle equivalence for logic optimization. In Section 8 we discuss what else needs to be done to make logic synthesis preserving high-level specification possible. Finally, we draw some conclusions in Section 9.

2. Toggle equivalence of Boolean functions

In this section, we introduce the notion of toggle equivalence. We also show that toggle equivalent Boolean functions can be considered as different implementations of the same multi-valued function.

2.1 Toggle equivalence of functions with identical sets of variables

Definition 1. Let $\boldsymbol{f_1}$:$\{0,1\}^n \rightarrow \{0,1\}^m$ and $\boldsymbol{f_2}$ $\{0,1\}^n \rightarrow \{0,1\}^k$ be m-output and k-output Boolean functions of the same set of variables. Functions $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are called ***toggle equivalent*** if $\boldsymbol{f_1}(\boldsymbol{x}) \neq \boldsymbol{f_1}(\boldsymbol{x'}) \Leftrightarrow \boldsymbol{f_2}(\boldsymbol{x}) \neq \boldsymbol{f_2}(\boldsymbol{x'})$. Circuits N_1 and N_2 implementing toggle equivalent functions $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are called ***toggle equivalent circuits***.

Informally, toggle equivalence means that for any pair of input vectors $\boldsymbol{x}$, $\boldsymbol{x'}$ for which at least one output of $\boldsymbol{f_1}$ "toggles", the same is true for $\boldsymbol{f_2}$ and vice versa.

Definition 2. Let $\boldsymbol{f}$ be a multi-output Boolean function of n variables. Denote by $Part(\boldsymbol{f})$ the partition of the set $\{0,1\}^n$ into disjoint subsets $B_1,\ldots,B_k$ such that $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x'})$ iff $\boldsymbol{x}$, $\boldsymbol{x'}$ are in the same subset B_i.

Proposition 1. Two Boolean functions $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are toggle equivalent iff $Part(\boldsymbol{f_1})$=$Part(\boldsymbol{f_2})$ i.e. iff for each element B_i of the partition $Part(\boldsymbol{f_1})$ there is an element B'_j of the partition $Part(\boldsymbol{f_2})$ such that B_i=B'_j and vice versa.

Proof. If $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are toggle equivalent, there cannot be a pair of vectors $\boldsymbol{x}$, $\boldsymbol{x'}$ such that $\boldsymbol{x}$, $\boldsymbol{x'}$ are in the same subset of one partition and in different subsets of the other partition. (Because that would mean that one function produces two identical output assignments while the other function toggles.)

Proposition 2. Let $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ be toggle equivalent single output Boolean functions. Then $\boldsymbol{f_1}$=$\boldsymbol{f_2}$ or $\boldsymbol{f_1}$=~$\boldsymbol{f_2}$ where '~' means negation.

Proof. From Proposition 1 it follows that $Part(\boldsymbol{f}_1)=Part(\boldsymbol{f}_2)$. Since $\boldsymbol{f}_1$, $\boldsymbol{f}_2$ are single output Boolean functions, $Part(\boldsymbol{f}_1)$ and $Part(\boldsymbol{f}_2)$ consist of two elements each. So $\boldsymbol{f}_1=\boldsymbol{f}_2$ or $\boldsymbol{f}_1=\sim\boldsymbol{f}_2$.

Definition 3. A multi-output Boolean function $\boldsymbol{f}$ of multi-valued variables is called an ***implementation*** of a multi-valued function F, if the truth table of $\boldsymbol{f}$ can be obtained from that of F in the following two steps

1) Replace the values of multi-valued variables of F with their codes (we assume that different values of a variable are assigned different codes);

2) Fill in the empty rows (if any) of the truth table with arbitrary Boolean vectors.

Proposition 3. Let $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ be toggle equivalent. Then $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are two different implementations of the same multi-valued function of Boolean variables.

Proof. According to Proposition 1, $Part(\boldsymbol{f}_1)=Part(\boldsymbol{f}_2)$. Let $Part(\boldsymbol{f}_1)$, $Part(\boldsymbol{f}_2)$ consist of k elements each. Then $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are implementations of the function F: $\{0,1\}^n \rightarrow \{1,..,k\}$ where $F(\boldsymbol{x})=m$, iff $\boldsymbol{x} \in B_m$, i.e. iff $\boldsymbol{x}$ is in the m-th element $Part(\boldsymbol{f}_1)$.

Proposition 3 is of great importance because it shows how one can reencode multi-valued variables **implicitly**. Suppose, a multi-output circuit N_1 implements a multi-valued function F and we want to reencode F. Synthesizing a circuit N_2 that is toggle equivalent to N_1 we obtain a new implementation of F. That is we reencode the output multi-valued variable of F not even knowing the binary encodings used to obtain N_1 and N_2 from F.

2.2 Toggle equivalence of functions with different sets of variables

In this subsection, the notion of toggle equivalence is extended to the case of Boolean functions with different sets of variables that are related by constraint functions.

Definition 4. Let X and Y be two disjoint sets of Boolean variables (the number of variables in X and Y may be different). A function $Cf(X,Y)$ is called a ***correlation function*** if there are subsets $A^X \subseteq \{0,1\}^{|X|}$ and $A^Y \subseteq \{0,1\}^{|Y|}$ such that

1) $|A^X| = |A^Y|$ and

2) $Cf(X,Y)$ specifies a bijective mapping M: $A^X \rightarrow A^Y$. Namely $Cf(\boldsymbol{x},\boldsymbol{y})=1$ iff $\boldsymbol{x} \in A^X$ and $\boldsymbol{y} \in A^Y$ and $\boldsymbol{y} = M(\boldsymbol{x})$.

Remark 1. Informally, $Cf(X,Y)$ is a correlation function if it specifies a bijective mapping between a subset A^X of $\{0,1\}^{|X|}$ and a subset A^Y of $\{0,1\}^{|Y|}$. So one can view $Cf(X,Y)$ as relating two different encodings of a $|A^X|$ – valued variable.

As Proposition 4 below shows, one can check if a Boolean function $H(X,Y)$ is a correlation one without explicitly finding subsets A^X and A^Y.

Proposition 4. Let X and Y be two disjoint sets of Boolean variables. A Boolean function $H(X,Y)$ is a correlation one iff the following two conditions hold:

1). There do not exist three vectors $\boldsymbol{x}$, $\boldsymbol{x}'$, $\boldsymbol{y}$ (where $\boldsymbol{x}$, $\boldsymbol{x}'$ are assignments to variables X and $\boldsymbol{y}$ is an assignment to variables Y) such that $\boldsymbol{x} \neq \boldsymbol{x}'$ and $H(\boldsymbol{x},\boldsymbol{y})=H(\boldsymbol{x}',\boldsymbol{y})=1$.

2) There do not exist three vectors $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{y}'$ such that $\boldsymbol{y} \neq \boldsymbol{y}'$ and $H(\boldsymbol{x},\boldsymbol{y})=H(\boldsymbol{x},\boldsymbol{y}')=1$.

Proof. Only if part. If $H(X,Y)$ is a correlation function, the fact that conditions 1) and 2) hold, follows from Definition 4.

If part. If conditions 1) and 2) hold then, $H(X,Y)$ specifies a bijective mapping between subsets A^X and A^Y defined in the following way. Subset A^X consists of all the assignments $\boldsymbol{x} \in \{0,1\}^{|X|}$ such that $H(\boldsymbol{x},\boldsymbol{y})=1$ for some $\boldsymbol{y} \in \{0,1\}^{|Y|}$. Subset A^Y consists of all the assignments $\boldsymbol{y} \in \{0,1\}^{|Y|}$ such that $H(\boldsymbol{x},\boldsymbol{y})=1$ for some $\boldsymbol{x} \in \{0,1\}^{|X|}$.

Remark 2. Checking if $H(X,Y)$ is a correlation function reduces to two satisfiability checks. Checking condition 1) of Definition 4 reduces to testing the satisfiability of the expression $H(X,Y) \wedge H(X',Y') \wedge Neq(X,X') \wedge Eq(Y,Y')$. Here $H(X',Y')$ is a "copy" of $H(X,Y)$ where variables of X',Y' are independent of those of X,Y. $Neq(\boldsymbol{x},\boldsymbol{x}')$ is equal to 1 iff $\boldsymbol{x} \neq \boldsymbol{x}'$. Function $Eq(Y,Y')$ is the negation of $Neq(Y,Y')$. Checking condition 2) of Definition 4 reduces to testing the satisfiability of $H(X,Y) \wedge H(X',Y') \wedge Eq(X,X') \wedge Neq(Y,Y')$. If both expressions are constant 0, then H is a correlation function.

Our definition of correlation function is different from the one given in [6] but serves the same purpose of relating two encodings of a multi-valued variable.

Definition 5. Let $\boldsymbol{f}_1$:$\{0,1\}^n \rightarrow \{0,1\}^m$ and $\boldsymbol{f}_2$:$\{0,1\}^p \rightarrow \{0,1\}^k$ be m-output and k-output Boolean functions and X and Y specify their sets of Boolean variables where $|X|=n$ and $|Y|=p$. Let $D_{inp}(X,Y)$ be a Boolean function. Functions $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are called ***toggle equivalent under constraint function*** $D_{inp}(X,Y)$ if $(\boldsymbol{f}_1(\boldsymbol{x}) \neq \boldsymbol{f}_1(\boldsymbol{x}') \wedge (D_{inp}(\boldsymbol{x},\boldsymbol{y})= D_{inp}(\boldsymbol{x}',\boldsymbol{y}')=1)) \Rightarrow (\boldsymbol{f}_2(\boldsymbol{y}) \neq \boldsymbol{f}_2(\boldsymbol{y}')$ and vice versa $(\boldsymbol{f}_2(\boldsymbol{y}) \neq \boldsymbol{f}_2(\boldsymbol{y}') \wedge (D_{inp}(\boldsymbol{x},\boldsymbol{y})=D_{inp}(\boldsymbol{x}',\boldsymbol{y}')=1)) \Rightarrow \boldsymbol{f}_1(\boldsymbol{x}) \neq \boldsymbol{f}_1(\boldsymbol{x}')$.

Proposition 5. Let X,Y be sets of Boolean variables and $\{X_1,..,X_s\}$ and $\{Y_1,..,Y_s\}$ be partitions of X and Y respectively. Let $Cf(X_1,Y_1),..,Cf(X_s,Y_s)$ be correlation functions. Let $\boldsymbol{f}_1(X)$ and $\boldsymbol{f}_2(Y)$ be toggle equivalent under the constraint function $D_{inp}(X,Y)=Cf(X_1,Y_1) \wedge \ldots \wedge Cf(X_s,Y_s)$. Then $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are implementations of the same multi-valued function of s multi-valued variables.

Proof follows from Proposition 3 and Remark 1.

2.3 Testing toggle equivalence

In this subsection, we show how one can check if multi-output Boolean circuits N_1 and N_2 are toggle equivalent. Namely we show that checking the toggle equivalence of N_1 and N_2 reduces to testing if function $D_{out}(N_1, N_2)$ specified by Definition 8 (see below) is a correlation one. This test can be performed by two satisfiability checks as described in Remark 2.

Definition 6. Let N be a Boolean circuit. Denote by $v(N)$ be the set of Boolean variables associated either with the output of a gate or a primary input of N. Denote by $\boldsymbol{Sat(v(N))}$ the Boolean function such that $Sat(z)=1$ iff the assignment z to variables $v(N)$ is "possible" i.e consistent. For example, if circuit N consists of just one AND gate $y=x_1 \wedge x_2$, then $v(N)=\{y, x_1,x_2\}$ and $Sat(v(N))=(\sim x_1 \vee \sim x_2 \vee y) \wedge (x_1 \vee \sim y) \wedge (x_2 \vee \sim y)$.

Definition 7. Let f be a Boolean function. We will say that function f^* is obtained from f by ***existentially quantifying away*** variable x if $f^* = f(\ldots,x=0,\ldots) \vee f(\ldots,x=1,\ldots.)$.

Definition 8. Let N_1 and N_2 be Boolean circuits whose inputs are specified by set of variables X and Y respectively. Let $D_{inp}(X,Y)$ be a Boolean function. Denote by $\boldsymbol{D_{out}(N_1, N_2)}$ the Boolean function obtained from the Boolean function H, where $H=Sat(v(N_1)) \wedge Sat(v(N_2)) \wedge D_{inp}(X,Y)$, by existentially

quantifying away all the variables of H but the output variables of N_1 and N_2.

Proposition 6. Let N_1 and N_2 be Boolean circuits with input variables specified by sets X, Y respectively. Let $D_{inp}(X,Y)$ be a Boolean function relating X and Y. Let $D_{inp}(X,Y)$ be a correlation function. Then N_1 and N_2 are toggle equivalent under constraint function $D_{inp}(X,Y)$ iff the function $D_{out}(N_1,N_2)$ specified in Definition 8 is also a correlation function.

Proof. Only If part. Let N_1 and N_2 be toggle equivalent. Then $D_{out}(N_1,N_2)$ satisfies either condition of Proposition 4 and hence it is a correlation function. For example, there cannot exist Boolean vectors $\boldsymbol{z}$, $\boldsymbol{z'}$ and $\boldsymbol{h}$ (where $\boldsymbol{z} \neq \boldsymbol{z'}$ and $\boldsymbol{z}, \boldsymbol{z'}$ are output assignments of N_1 and $\boldsymbol{h}$ is an output assignment of N_2) such that $D_{out}(\boldsymbol{z},\boldsymbol{h})=D_{out}(\boldsymbol{z'},\boldsymbol{h})=1$. Indeed, it would mean that there exist pairs of vectors $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{x'}, \boldsymbol{y'}$ such that a) $\boldsymbol{z}=N_1(\boldsymbol{x})$, $\boldsymbol{z'}= N_2(\boldsymbol{x'})$ and $\boldsymbol{h} = N_2(\boldsymbol{y})=N_2(\boldsymbol{y'})$;b) $D_{inp}(\boldsymbol{x}, \boldsymbol{y})=1$ and $D_{inp}(\boldsymbol{x'}, \boldsymbol{y'})=1$; c) $\boldsymbol{x} \neq \boldsymbol{x'}$ and $\boldsymbol{y} \neq \boldsymbol{y'}$; d) $N_1(\boldsymbol{x}) \neq N_1(\boldsymbol{x'})$ while $N_2(\boldsymbol{y}) = N_2(\boldsymbol{y'})$. But this is impossible since N_1 and N_2 are toggle equivalent.

If part can be proven in a similar manner.

3. Common specification and toggle equivalence

In this section, we show that the existence of a CS of single output combinational circuits N_1 and N_2 means that N_1, N_2 can be partitioned into toggle equivalent subcircuits that are connected in N_1 and N_2 "in the same way". The main result of this section is formulated in Proposition 7.

Definition 9. Let $N = (V,E)$ be a DAG representing a Boolean circuit (here V,E are sets of nodes and edges of N respectively.) A subgraph $N^*=(V^*, E^*)$ of N is called a ***subcircuit*** if the following two conditions hold:

a) if g_1, g_2 are in V^* and there is a path from g_1 to g_2 in N, then all the nodes of N that on that path are in V^* ;

b) if g_1, g_2 of V^* are connected by an edge in N, then they are also connected by an edge in N^*.

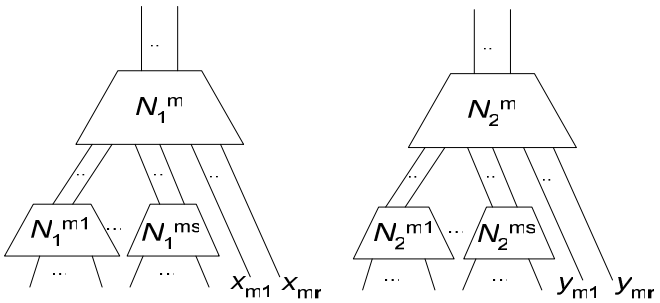

Figure 2. Illustration to Definition 12

Definition 10. Let N^* be a subcircuit of N. An input of a gate g of N^* is called ***an input*** of N^* if it is not connected to the output of some other gate of N^*. The output of a gate g is called ***an output*** of subcircuit N^* if a) it is the primary output of N; b) it is connected to an input of a gate of N that is not in N^*.

Definition 11. Let a Boolean circuit N be partitioned into k subcircuits N^1, ..., N^k . Let T be a directed graph of k nodes such that nodes G_i and G_j of T are connected by a directed edge (from n_i to n_j) iff an output of N^i is connected to an input of N^k in N. T is called ***the communication specification*** corresponding to the partition N^1, ..., N^k . The partition N^1, ..., N^k is called ***topological*** if T is a DAG (i.e. if T does not contain cycles)

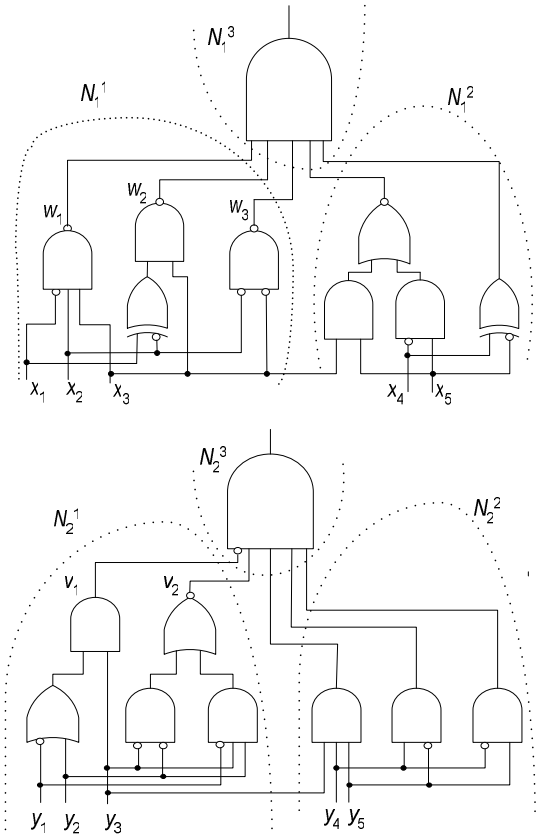

Figure 3. Example of circuits N_1 and N_2 with a CS

Definition 12. Let T be the communication specification of circuit N with respect to a topological partition N^1, ..., N^k. Let G_i be the node of T corresponding to subcircuit N^i. The length of the longest path from an input of T to G_i is called the ***level*** of G_i and N^i (denoted by $level(G_i)$ and $level(N^i)$ respectively).

Definition 13. Let N_1^1, ...,N_1^k and N_2^1, ..., N_2^k be topological partitions of single output Boolean circuits N_1, N_2. Let communication specifications of N_1 and N_2 with respect to partitions N_1^1, ..., N_1^k and N_2^1, ..., N_2^k be identical. Denote by $\boldsymbol{D_{out}(N_1^m,N_2^m)}$, $\boldsymbol{m=1,\ldots,k}$ functions computed by induction in topological levels. Namely, we first compute functions D_{out} for the subcircuits of level 1, then for those of level 2 and so on. Function $D_{out}(N_1^m, N_2^m)$ is obtained from function $H=Sat(v(N_1^m)) \wedge Sat(v(N_2^m)) \wedge D_{inp}(N_1^m, N_2^m)$ by existentially quantifying away all the variables except the output variables of N_1^m, N_2^m. The function $D_{inp}(N_1^m, N_2^m)$ is equal to $D_{out}(N_1^{m1}, N_2^{m1}) \wedge \ldots \wedge D_{out}(N_1^{ms}, N_2^{ms}) \wedge Eq(x_{m1}, y_{m1}) \wedge\ldots\wedge Eq(x_{mr}, y_{mr})$. Here N_1^{m1}, ,.., N_1^{ms}, N_2^{m1},.., N_2^{ms} are the s subcircuits (if any) whose outputs are connected to inputs of N_1^m, N_2^m respectively. (See illustration in Figure 2.) Variables $x_{m1},\ldots, x_{mr}$, $y_{m1},.., y_{mr}$ are the r primary input variables of N_1 and N_2 (if any) that feed N_1^m and N_2^m respectively. Function $Eq(x_{mt},y_{mt})$, $1 \le t \le r$ is equal to 1 iff x_{mit} is equal to y_{mt}.

Proposition 7. Let N_1, N_2 be two functionally equivalent single output circuits. Let T be a DAG of k nodes. Circuits N_1 and N_2 are implementations of a specification S whose topology is given by T iff there is a partitioning $Spec(N_1) = \{N_1^1, ..., N_1^k\}$ of N_1 and a partitioning $Spec(N_2) = \{N_2^1, ..., N_2^k\}$ of N_2 into k subcircuits such that

a) Communication specifications T_1,T_2 of N_1 and N_2 with respect to partitionings $Spec(N_1)$, $Spec(N_2)$ are equal to T;

b) Each pair of circuits N_1^m, N_2^m is toggle equivalent under constraint function $D_{inp}(N_1^m, N_2^m)$ specified by Definition 12.

Sketch of the proof. If part. It is proven by induction (in levels) using Proposition 5 and Proposition 6. ***Only if part*** is proven by induction using the fact that two Boolean functions implementing the same multi-valued function are toggle equivalent.

Example. An example of circuits with a CS of three blocks is shown in Figure 3. Circuit N_1 (at the top) and N_2 (at the bottom) have the same communication specification (shown in Figure 1 on the left side). Subcircuits N_1^1, N_1^2 (outlined by the dotted line) are toggle equivalent to subcircuits N_2^1, N_2^2 respectively in terms of their inputs related by the constraint functions $D_{inp}(N_1^1, N_2^1) = Eq(x_1, y_1) \wedge Eq(x_2, y_2) \wedge Eq(x_3, y_3)$ and $D_{inp}(N_1^2, N_2^2) = Eq(x_3, y_3) \wedge Eq(x_4, y_4) \wedge Eq(x_5, y_5)$ respectively.

Consider, for example subcircuits N_1^1 and N_2^1. For the pair of input assignments (x_1=0, x_2=1, x_3=0) and (x_1=0, x_2=1, x_3=1) the outputs of N_1^1 take values (w_1=1,w_2=1,w_3=1) and (w_1=0,w_2=1,w_3=1) respectively i.e. N_1^1 toggles. For the pair of the corresponding input assignments (y_1=0, y_2=1, y_3=0) and (y_1=0, y_2=1, y_3=1) the outputs of N_2^1 take values (v_1=0,v_2=1) and (v_1=1,v_2=0) respectively. So N_2^1 toggles as well. On the hand for the pair of input assignments (x_1=0, x_2=0, x_3=0) and (x_1=1, x_2=0, x_3=0) the outputs of N_1^1 take the same assignment w_1=1,w_2=1,w_3=0 and hence N_1^1 does not toggle. For the corresponding pair of input assignments (y_1=0, y_2=0, y_3=0) and (y_1=1, y_2=0, y_3=0) the outputs of N_2^1 take the same value (v_1=0,v_2=0). So N_2^1 does not toggle either.

It is not hard to check that subcircuits N_1^3 and N_2^3 are toggle equivalent in terms of their local inputs related by the constraint function $D_{inp}(N_1^3, N_2^3) = D_{out}(N_1^1, N_2^1) \wedge D_{out}(N_1^2, N_2^2)$. The functions $D_{out}(N_1^1, N_2^1)$, $D_{out}(N_1^2, N_2^2)$ are obtained as described in Definition 13.

4. A Procedure for checking if a common specification of two circuits is correct

In this section, we describe a procedure for checking if a predefined CS of circuits N_1,N_2 is correct. We will refer to this procedure as ***Common Specification Verification* (*CSV*)**. As a "by-product", our CSV procedure checks if N_1,N_2 are functionally equivalent.

The EC procedure for circuits N_1, N_2 with a CS introduced in [6] and modified in [7] essentially uses an implicit representation of this CS as partitions of N_1 and N_2. Proposition 7 allows one to modify this procedure so that in addition to EC it also checks the correctness of the CS. The pseudocode of the CSV procedure is shown in Figure 4.

The procedure *topol_partition* checks if $Spec(N_1)$ and $Spec(N_1)$ are topological partitions (see Definition 11). The procedure *equiv_commun_specs* checks if communication specifications T_1 of N_1 with respect to $Spec(N_1)$ and T_2 of N_2 with respect to $Spec(N_2)$ are identical.

In the main loop, functions $D_{out}(N_1^i,N_2^i)$ are computed in topological order as described in Definition 12. Before computing $D_{out}(N_1^i,N_2^i)$ the procedure *constr_func* forms the expression D_{inp} (see Definition 13).

```
/* --- Spec(N1)= {N1^1,..,N1^k},Spec(N2)= {N2^1,..,N2^k} ---*/
CSV(N1, N2, Spec(N1),Spec(N2)) {
 if (topol_partition(N1,N2,Spec(N1),Spec(N2)) == 'no')
     return('CS_check_failure');

  if (equiv_commun_specs( N1,N2,Spec(N1),Spec(N2)) == 'no')
       return('CS_check_failure');

 for (i=1; i <= k ; i++) {
   D_inp = constr_func(N1^i,N2^i,N1,N2);
    D_out(N1^i, N2^i) = exist_quantify(N1^i,N2^i, D_inp);
    if (correlation_function(D_out) == 'no')
        return('CS_check_failure');}

 if (D_out(N1^k, N2^k) implies equivalence_ function)
       return('equivalent');
 else
       return('CS_check_failure');}
```

Figure 4. Pseudocode of the CSV procedure

The function *exist_quantify* existentially quantifies away from the function $H=Sat(v(N_1^m)) \wedge Sat(v(N_2^m)) \wedge D_{inp}$ all the variables except the output variables of N_1^i and N_2^i. Then the *correlation_function* procedure checks if the result of quantification D_{out} is a correlation function. The check is performed as described in Remark 2.

Finally, the CSV procedure checks if the correlation function of subcircuits N_1^k and N_2^k (whose primary outputs are primary outputs of N_1 and N_2) implies the equivalence function $Eq(y, z)$. (Here y, z are Boolean variables associated with the outputs of N_1 and N_2 respectively). If so, then N_1 and N_2 are declared equivalent. Otherwise, the CSV procedure returns the '*CS_check_failure*' answer. This answer is also returned if any of the checks performed by *topol_partition*, *equiv_commun_specs* and *correlation_function* fails.

Definition 14. Let N be a circuit with a specification S represented by partition $Spec(N)=\{N^1..,N^k\}$. The ***granularity*** of specification S for N is the size (i.e. the number of gates) of the largest subcircuit N^j, j=1,..,k.

Definition 15. Let N_1 and N_2 be implementations of the same specification S. Let p_1 and p_2 be granularities of specification S for N_1 and N_2 respectively. We will call the maximum of p_1 and p_2 the ***granularity of the CS S*** of N_1, N_2.

The CSV procedure is exponential in the granularity p of CS S of N_1, N_2 and is linear in the number of blocks of S i.e. in the number of subcircuits in $Spec(N_1)$, $Spec(N_2)$. The exponentiality in p is due to procedures *exist_quantify* and *correlation_function*. The reason why the CSV procedure is exponential only in p and not in the circuit size is that the two exponential procedures above are applied only to subcircuits N_1^i, N_2^i whose size is bounded by p. Suppose that the value of p is bounded by a constant (i.e. circuits N_1, N_2 can be of arbitrary size but the granularity of their CS is bounded). Then the CSV procedure proves the equivalence of

specifications *Spec*(N_1) and *Spec*(N_2) (and hence functional equivalence of N_1 and N_2) in **linear time** in the circuit size.

5. Equivalence checking of circuits with unknown specification

Note that the efficiency of our CSV procedure is due to the fact that a CS specification of N_1 and N_2 (represented by *Spec*(N_1) and *Spec*(N_2)) is known. A natural question to ask is as follows. Suppose circuits N_1, N_2 have a CS specification S of small granularity p. Is there an efficient procedure for EC of N_1, N_2 if S is unknown (i.e we do not know the partitions *Spec*(N_1) and *Spec*(N_2) representing S)? In [6][5] it was conjectured that in that case EC of N_1,N_2 is hard for any deterministic algorithm. The new (and equivalent) definition of CS given in this report allows one to get a better perspective on the problems one has to solve when checking N_1,N_2. for equivalence.

One way to do the job is to find *Spec*(N_1) and *Spec*(N_2) and apply the CSV procedure. This approach is very similar to what the existing EC procedures exploiting structural similarity of N_1, N_2 do ([1][2][3][8][9][11]). Namely, they try to find pairs of functionally equivalent points of N_1,N_2 and use them as cut points. Then new points of N_1,N_2 that are functionally equivalent in terms of cut points are looked for. The idea is that checking functional equivalence of internal points of N_1,N_2 in terms of cut points is much easier than in terms of primary inputs. This approach faces the following two problems. The first problem is to find new potential cut points (i.e. to find points of N_1,N_2 that are functionally equivalent). The second problem is to decide whether two functionally equivalent internal points can be used as cut points. Making a wrong decision here leads to the appearance of so called "false negatives".

One can view the "cut advancement" approach above as search for a CS of N_1,N_2 of a special type where every subcircuit of *Spec*(N_1) and *Spec*(N_2) has exactly one output. However, if one try to extend this approach to CSs of the general type (where subcircuits of *Spec*(N_1) and *Spec*(N_2) may have many outputs), the two problems mentioned above become virtually unsolvable. In the case of multi-output subcircuits, functional equivalence is replaced with toggle equivalence. Let the granularity p of a CS of N_1, N_2 be equal to 10. (So the subcircuits of *Spec*(N_1) and *Spec*(N_2) may have up to 10 outputs.) Then the number of candidate subcircuits in N_1 and N_2 is proportional to $|N_1|^{10}$ and $|N_2|^{10}$ respectively where $|N_j|$ is the size of N_j. The number of potential pairs of subcircuits to examine is proportional to $|N_1|^{10}*|N_2|^{10}$. But even if one finds subcircuits N_1^i, N_2^i of size less or equal to 10 that are toggle equivalent, one still needs to decide if the outputs of N_1^i,N_2^i can be used as cut points. That is one needs to decide whether N_1^i, N_2^i are toggle equivalent "by chance" or they are a part of a CS. Since the number of candidates is huge, making a mistake becomes unavoidable.

One can also try to perform EC of N_1 and N_2 by a procedure like recursive learning [10] that does not need the knowledge of a CS. The problem is that to prove N_1 N_2 to be equivalent, one needs to derive relations between $2*p$ Boolean variables. If a CS of N_1, N_2 is not known, the number of relations one needs to derive in the worst case is proportional to $(|N_1|+|N_2|)^{2*p}$, which makes such a procedure computationally infeasible.

6. On logic synthesis of circuits preserving predefined specification

In this section we describe a procedure that, given a circuit N_1 with a known specification, builds another circuit N_2 implementing the same specification as N_1.

Let N_1 be a Boolean circuit that needs to be optimized. Let S be a specification of N_1 represented as *Spec*(N_1)= $\{N_1^1,..,N_1^k\}$. Figure 5 shows pseudocode of a procedure for generating a circuit N_2 that implements the same specification as circuit N_1. We will refer to it as ***Specification Preserving*** (***SP***) procedure. We assume that subcircuits $N_1^1,..,N_1^k$ are numbered in topological order i.e. for every pair i,j such that $i < j \Rightarrow level(N_1^i) \le level(N_1^j)$.

Synthesize(N_1, *Spec*(N_1),*cost_functions*) {
 for (i=1; i <= k ; i++) {
  D_{inp}= *constr_func*(N_1^i,N_2^i,N_1,N_2);
  N_2^i = *synth_toggle_equivalent*(N_1^i, D_{inp},*cost_functions*)
  $D_{out}(N_1^i, N_2^i)$ = *exist_quantify*(N_1^i,N_2^i, D_{inp}); }
return(N_2,*Spec*(N_2))}

Figure 5. Pseudo code of the SP procedure

The idea of the SP procedure is to replace subcircuits $\{N_1^1,.., N_1^k\}$ with toggle equivalent subcircuits $\{N_2^1,.., N_2^k\}$ in topological order moving from inputs to outputs. The SP procedure returns circuit N_2 implementing the same specification as N_1. Circuit N_2^i toggle equivalent to N_1^i is built by the *synth_toggle_equivalent* procedure. (Section 7 gives an example of such a procedure.) After N_2^i is synthesized we compute the correlation function $D_{out}(N_1^i,N_2^i)$ using previously computed functions D_{out} exactly as it is done by the CSV procedure. (Note that since N_1^i, N_2^i are toggle equivalent "by construction", $D_{out}(N_1^i,N_2^i)$ is a correlation function.)

The importance of the SP procedure is twofold. First, the complexity of the SP procedure is the same as that of the CSV procedure. Namely, it is exponential in the granularity p of the CS of N_1, N_2 represented by *Spec*(N_1),*Spec*(N_2) and linear in the number of subcircuits in *Spec*(N_1) and *Spec*(N_2). (Here we make a realistic assumption that *synth_toggle_equivalent* is "only" exponential in p). This means that if p is fixed, the SP procedure is linear in circuit size and hence it is **scalable**.

Second, the SP procedure allows one to make a nice **trade-off** between optimization quality and efficiency. Note that the search space explored by the SP procedure is limited to the implementations of the specification of N_1 represented by *Spec*(N_1). The smaller the granularity of specification S of N_1 is, the smaller the search space is, which implies greater efficiency of the SP procedure. So, if no good alternative implementation N_2 of S is found for the current specification of N_1, one can merge some adjacent subcircuits of *Spec*(N_1) to get a specification with a larger value of granularity for N_1 . This way the search space becomes larger at the expense of performance degradation of the SP procedure.

7. Experimental Results

In this section, we give some experimental results. In Subsection 7.1 we show that equivalence checking even of very similar circuits N_1 N_2 (i.e. circuits having a CS of small granularity) is hard if this CS is not known. In Subsection 7.2 we use MCNC benchmarks to show that one can optimize a medium

size circuit N_1 by removing logical redundancy and obtaining another (smaller) circuit N_2 that is toggle equivalent to N_1.

7.1 Equivalence checking

In the experiments we compared the performance of two EC algorithms: our CSV procedure shown in Figure 4 and an Industrial Equivalence Checker (referred to as **IEC**) of very high quality. Both algorithms were run on a 3.06 GHz Xeon PC.

In the experiments we checked for equivalence circuits obtained from a specification given as a combinational circuit of multi-valued blocks. The number of values taken by the variables of a block was parameterized. Circuits N_1, N_2 to be checked for equivalence were obtained from a specification using two sets of random encodings of the minimum (logarithmic) length.

Table 1. EC of circuits obtained from 4-valued specifications

Name	# blocks in CS	CSV (sec.)	IEC (sec.)	Ratio (IEC / CSV)
des1	705	0.4	3	**7**
des2	2,562	2	14	**7**
des3	3,519	3	281	**94**
des4	8,628	14	308	**22**
des5	9,027	16	543	**34**
des6	10,572	20	534	**27**

The goal of experiments was twofold. First we wanted to show that EC of circuits with a CS S of even small granularity is hard if S is unknown. Second, we wanted to demonstrate that this weakness of current EC algorithms hinders the development of more powerful synthesis procedures. (Even though we obtained circuits N_1, N_2 by explicitly encoding multi-valued variables of specification, circuit N_2 could have been obtained from N_1 by the synthesis procedure described in Section 6.)

In Table 1 we consider specifications with blocks of 4-valued variables. Second column gives the number of blocks for each design. Third and fourth columns give runtimes for CSV and IEC. The last column gives the ratio of runtimes.

Table 2. EC of circuits obtained from 8-valued specifications

Name	# blocks in CS	CSV (sec.)	IEC (sec.)	Ratio (IEC / CSV)
des1	705	5	16,948*	**> 3,390**
des2	2,562	19	24,638*	**> 1,297**
des3	3,519	29	>36,000	**> 1,241**
des4	8,628	107	26,758	**250**
des5	9,027	111	>36,000	**> 324**
des6	10,572	141	27,391	**194**

In Table 2 we consider the same specifications (i.e. the topology of corresponding specifications was the same) of 8-valued blocks. Hence *the granularity of CSs* of binary circuits obtained by encoding multi-valued variables was slightly larger than for binary circuits of Table 1. Runtimes of IEC marked with '*' correspond to the cases where IEC aborted without completion (due to exhausting some internal resource).

For the circuits from both tables CSV was faster than IEC. However the gap between the performance of CSV and IEC increased dramatically as the granularity of CSs had grown. IEC was able to complete all the instances of Table 1 in a reasonable time. On the other hand, it completed only 2 equivalence checks for the circuits of Table 2 and took dramatically more time.

7.2 Toggle equivalence based redundancy removal

The goal of experiments described in this subsection was twofold. First, we wanted to show that one can efficiently check toggle equivalence of two practical circuits of medium size. Second, we wanted to demonstrate that one can use the notion of toggle equivalence for logic optimization.

In Section 6 we described a method of logic synthesis that preserves a predefined specification. The key procedure of the algorithm shown in Figure 5 is *synth_toggle_equivalent*. Given a subcircuit N_1^i and a cost function, this procedure builds another subcircuit N_2^i that is toggle equivalent to N_1^i and is optimized with respect to this cost function. In this subsection we give an example of such a procedure. This procedure is based on stuck-at fault redundancy removal. Suppose that N_1 is a multi-output circuit to be optimized (for the sake of simplicity, in this section we drop superscripts from the symbols denoting subcircuits with the exception of the last few paragraphs). Suppose that N_2 is the circuit obtained from N_1 by setting to a constant $a \in \{0,1\}$ the line connecting the output of a gate g_i of N_1 to an input of gate g_k of N_1. Suppose N_2 and N_1 are functionally equivalent. This means that one can remove the connection between gates g_i and g_k and set the corresponding input of g_k to the constant a without changing the functionality of N_1 (which means that N_1 has some logic redundancy).

Suppose however that we relax the requirement of preserving the functional equivalence of N_1 and N_2. In other words, suppose that after setting the output of the gate g_k to a constant (as described above) we get a circuit N_2 that is toggle equivalent to N_1. Setting a line to a constant can be considered as an example of transformations that can be used by the procedure *synth_toggle_equivalent* above. On the one hand, by removing redundancies that preserve toggle equivalence (but may break functional equivalence) we optimize circuit N_1. On the other hand we build a circuit that is toggle equivalent to N_1. Since functional equivalence is a special case of toggle equivalence, logic redundancy removal that preserves toggle equivalence is a more powerful optimization technique than its counterpart preserving functional equivalence.

In this subsection, we test logic redundancy of some MCNC benchmarks with respect to toggle equivalence. But first we show how one can check toggle equivalence of the original and faulty circuits. To check if circuits N_1 and N_2 are toggle equivalent one can use the method described in Proposition 6. Let us assume for the sake of clarity that N_1 and N_2 have the same set of input variables $X=\{x_1,..,x_n\}$. Then to check if N_1 and N_2 are toggle equivalent one can a) existentially quantify away from the function $H(N_1, N_2) = Sat(v(N_1) \wedge Sat(v(N_2))$ all the variables except the output variables of N_1 and N_2; b) check if the function obtained from H after existential quantification is a correlation function as described in Remark 2. However, it is not hard to see

that one can check N_1 and N_2 for toggle equivalence without existential quantification.

Let $Y=\{y_1,\ldots, y_m\}$ and $Z=\{z_1,.., z_k\}$ be the sets of output variables of N_1 and N_2 respectively. Then checking if N_1 and N_2 are toggle equivalent reduces to two SAT checks similar to those of Remark 2. The first check is to test if the function $H_1 = H(N_1, N_2) \wedge H(N^*_1, N^*_2) \wedge Neq(Y,Y^*) \wedge Eq(Z,Z^*)$ is satisfiable. Here N^*_1 and N^*_2 are copies of circuits N_1 and N_2, with input variables represented by $X^*=\{x^*_1,.., x^*_n\}$ and their output variables represented by $Y^*=\{y^*_1,\ldots, y^*_m\}$ and $Z^*= \{z^*_1,.., z^*_k\}$ respectively (see Figure 6). The value of $Eq(\boldsymbol{z}, \boldsymbol{z^*})$ where $\boldsymbol{z}$ and $\boldsymbol{z^*}$ are assignments to Z and Z^* respectively is equal to 1 iff $\boldsymbol{z}=\boldsymbol{z^*}$. The function $Neq(Y, Y^*)$ is the negation of $Eq(Y,Y^*)$. The second SAT check is to test if the function $H_2 = H(N_1, N_2) \wedge H(N^*_1, N^*_2) \wedge Eq(Y,Y^*) \wedge Neq(Z,Z^*)$ is satisfiable.

Circuits N_1 and N_2 are toggle equivalent iff H_1 and H_2 are unsatisfiable. For example, if N_1 is satisfiable, then there is a pair of assignments $\boldsymbol{x}$, $\boldsymbol{x^*}$ to variables X and X^* respectively such that N_1 and N^*_1 produce different output assignments while N_2 and N^*_2 produce the same assignment (which means that N_1 toggles and N_2 does not).

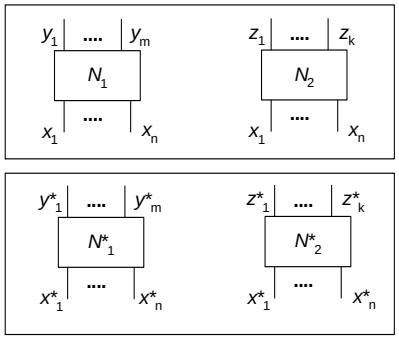

Figure 6. Two copies of N_1 and N_2 one needs for checking their toggle equivalence

Table 3 gives results of redundancy removal for MCNC benchmarks. In the first column the names of the MCNC benchmarks we used in experiments are shown. First, initial MCNC benchmarks were technology mapped to obtain circuits consisting of two input NAND gates. The technology mapping was performed by the "tech_decomp" command of the logic synthesis system SIS [10]. The size of obtained circuits (number of inputs, outputs and gates) is shown in columns 2-4. Then the circuits were optimized by removing stuck-at fault redundancy as described above. This optimization was performed by running the "red_removal" command of SIS. The size of optimized circuits is shown in the column "final number of gates" of Table 3. Note that the resulting circuits do not contain untestable stuck-at faults any more. The last column of Table 3 shows the number of untestable stuck-at faults with respect to toggle equivalence circuits still have after removing all the single stuck-at faults with respect to functional equivalence. This column shows that more than one-third of circuits still have stuck-at faults that are untestable with respect to toggle equivalence and some circuits (like vda, x1,K2) can be significantly optimized by removing this redundancy.

To check toggle equivalence of original and faulty circuits we ran two SAT checks as described above. The SAT checks were performed by the SAT-solver BerkMin [4] . Table 4 gives some

Table 3. Redundancy removal from MCNC benchmarks

name	#in-puts	#out-puts	ini-tial number of gates	final number of gates	number of red. faults w.r.t. toggle equi-valence
pcler8	27	17	86	86	**8**
frg1	28	3	792	792	0
sct	19	15	207	**202**	0
unreg	36	16	128	128	0
lal	26	19	198	198	**28**
c8	28	18	332	**235**	0
cht	47	36	374	**253**	0
b9	41	21	147	**141**	**10**
my_adder	33	17	256	256	0
example2	85	66	382	**328**	**35**
C432	36	7	218	**175**	0
apex7	49	37	327	**290**	0
vda	17	39	1333	1333	**2125**
ttt2	24	21	670	**387**	0
i5	133	66	423	423	0
i6	138	67	760	**717**	0
term1	34	10	854	**494**	0
i7	199	67	972	**893**	0
i9	88	63	1163	**812**	0
K2	45	43	2875	**2643**	**587**
apex6	135	99	747	747	**8**
x4	94	71	959	**737**	**6**
x3	135	99	1547	**1326**	0
x1	51	35	2140	**1913**	**318**
C499	41	32	446	**438**	0
rot	135	107	1359	**1193**	**85**
C880	60	26	360	360	**6**
frg2	143	139	2434	**1729**	**76**
C1355	41	32	550	**542**	0
pair	173	137	1916	**1596**	**38**

data on the time taken by those SAT checks. The column "max time" gives the maximum time taken by a SAT check when testing stuck-at fault redundancy (with respect to toggle equivalence) of a particular circuit. The column "Median time" gives the median time among all the SAT checks and the column "Arithmetic mean" gives the average time taken by BerkMin when testing redundancy of a particular circuit. The results of Table 4 show

that toggle equivalence can be efficiently checked by a state-of-the-art SAT-solver.

Table 4. Performance of Sat-solver in toggle equivalence checks

name	Max time	Median time	Arithmetic mean
pcler8	0.03	0.010	0.01
frg1	0.89	0.100	0.14
sct	0.16	0.040	0.04
unreg	0.08	0.010	0.01
lal	0.20	0.040	0.05
c8	0.42	0.040	0.07
cht	0.18	0.020	0.04
b9	0.20	0.030	0.04
my_adder	0.36	0.050	0.07
example2	0.56	0.100	0.13
C432	0.27	0.040	0.05
apex7	0.36	0.060	0.08
vda	3.23	0.680	0.64
ttt2	3.09	0.070	0.20
i5	0.89	0.270	0.30
i6	2.51	0.080	0.25
term1	1.47	0.120	0.19
i7	5.21	0.370	0.73
i9	3.02	0.170	0.40
K2	13.38	2.500	3.96
apex6	3.83	0.300	0.51
x4	2.38	0.270	0.49
x3	6.24	1.040	1.37
x1	11.84	1.850	2.56
C499	13.06	0.110	1.11
rot	7.94	1.110	1.71
C880	10.86	0.070	0.53
frg2	13.01	2.180	2.78
C1355	16.24	0.150	1.32
pair	17.10	1.810	2.91

Unfortunately, the method of obtaining toggle equivalent circuits by removing logic redundancy that preserves toggle equivalence is "incomplete". Suppose for example, that we want to optimize a circuit N_1 whose specification $Spec(N_1)=\{N_1^1, N_1^2, N_1^3\}$ is shown in Figure 1. Using the procedure of logic redundancy removal above, only circuits of the first topological level i.e. N_1^1, N_1^2 can be optimized by replacing them with their toggle equivalent counterparts N_2^1, N_2^2. To finish synthesis of circuit N_2 we have to compute correlation functions $D_{out}(N_1^1, N_2^1)$ and $D_{out}(N_1^2, N_2^2)$ and then synthesize a subcircuit N_2^3 that is toggle equivalent to N_1^3 under the constraint function $D_{out}(N_1^1, N_2^1) \wedge D_{out}(N_1^2, N_2^2)$. This last circuit cannot be obtained by simply removing redundant logic from the subcircuit N_1^3. However, results of this subsection imply that it is feasible to design an efficient procedure for building a subcircuit N_2^3 that is toggle equivalent to N_1^3.

8. Directions for future research

In this section we sketch three directions for future research. The first direction is to apply the results of the theory we introduced in this report to sequential circuits. It is of great practical importance because a sequential circuit has a "natural partitioning" which is a partitioning of this circuit into combinational subcircuits bounded by registers. The idea is that one can use the notion of toggle equivalence for encoding state variables implicitly. The procedure for equivalence checking of two sequential circuits is a straightforward generalization of the corresponding procedure for combinational circuits with a CS.

The second direction is to develop more powerful procedures for existential quantification which is a core operation for both equivalence checking of circuits with a known CS and logic synthesis preserving a predefined specification. Suppose that N is a k-output Boolean circuit and one needs to existentially quantify away all the variables of N except output variables. If the value of k is small one can perform 2^k SAT-checks if a particular output assignment is observable under some input assignment. Note, that such a SAT-based quantification can be performed even if the size of N is large. If k is large, then one can use BDDs to perform quantification. The idea is to represent the function $Sat(v(N))$ as a BDD and quantify away all the variables except output ones. The drawback of this method that it may occur that even though the set of observable output combinations is "reasonably regular" there is no any good ordering of output variables and so the final BDD is too large to compute. So if N has a regular set of observable output combinations (but there is no small BDD representing it) and k is large, no current method of existential quantification can compute this set.

The third direction is to find efficient and high-quality procedures to solve the following problem. Let N_1 be a multi-output Boolean circuit and X be its set of input variables. Let Y be another set of Boolean variables such that $X \cap Y \neq \varnothing$ and $Cf(X,Y)$ be a correlation function. The problem is to find a circuit N_2 with the set of input variables Y such that a) N_1 and N_2 are toggle equivalent under the constraint function $Cf(X,Y)$; b) N_2 is optimized with respect to a cost function. (For example, N_2 has fewer gates than N_1.) This operation is key to logic synthesis preserving a predefined specification.

9. Conclusions

In this report, we show that two combinational circuits N_1, N_2 have a CS S iff they can be partitioned into toggle equivalent subcircuits connected in N_1, N_2 in the same way. We give an efficient procedure for verifying a CS of N_1, N_2 that also performs EC of N_1, N_2. We show how one can build a combinational circuit that preserves a predefined specification. We give experimental evidence that EC of circuits with unknown CS is hard. Besides we experimentally show that the notion of toggle equivalence can be used for logic optimization of practical circuits.

References

[1] C.L. Berman, L.H.Trevillyan. *Functional comparison of logic designs for VLSI circuits.* ICCAD-89, pp.456-459.

[2] D.Brand. *Verification of large synthesized designs.* ICCAD-93,pp.534-537.

[3] J.R.Burch,V.Singhal. *Tight integration of combinational verification methods.* ICCAD-98, pp.570-576.

[4] E.Goldberg, Y.Novikov. *BerkMin: A Fast and Robust SAT-solver.* DATE-2002, Paris,pp.142-149..

[5] E.Goldberg,Y.Novikov. *How good can a resolution based SAT-solver be?* SAT-2003, LNCS 2919,pp.37-52.

[6] E.Goldberg, Y. Novikov. *Equivalence Checking of Dissimilar Circuits.* International Workshop on Logic and Synthesis, May 28-30, 2003, USA. Available at *http://eigold.tripod.com/papers/dissim-iwls.zip*

[7] E.Goldberg. *Equivalence Checking of Dissimilar Circuits II.* Technical report. CDNL-TR-2004-08030, August 2004, available at *http://eigold.tripod.com/papers/tr-2004-0830.pdf*

[8] C.van Eijk,G.Janssen. *Exploiting structural similarities in a BDD-based verification method.* Proceedings of 2nd International Conference on Theorem Provers in Circuit Design, pp.110-125,1995.

[9] A.Kuehlmann, F.Krohm. *Equivalence checking using cuts and heaps*, DAC-98, pp.263-268.

[10] W.Kunz, D.Pradhan. Recursive Learning: *A New Implication Technique for Efficient Solutions to CAD-problems: Test, Verification and Optimization.* IEEE transactions on CAD, Vol. 13, No. 9, pp. 1143-1158, 1994.

[11] Y.Matsunaga. *An efficient equivalence checker for combinatorial circuits.* DAC-96,pp.629-634.

[12] E. Sentovich et al. *SIS: A system for sequential circuit analysis*, tech. rep., Electronics Research Laboratory, University of California, Berkeley, May 1992.