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

Eugene Goldberg
Cadence Berkeley Labs
1995, University Ave., suite 460, Berkeley,CA, USA, 94704
tel. 1-510-647-2825, email: egold@cadence.com

ABSTRACT. In this paper we develop a theory of equivalence checking (EC) 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 circuits N_1, N_2 have the same predefined specification. As a "by-product", this procedure performs EC of N_1 and N_2. 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 EC of N_1, N_2 is hard if their CS is unknown.

Categories and Subject Descriptors
B.6.3 [**Logic Design**]: Design Aids – automatic synthesis, optimization, verification.

General Terms. Algorithms, Theory, Design,Verification

Keywords. common specification, scalable equivalence checking, scalable logic synthesis, toggle equivalence.

1. INTRODUCTION

In this paper we continue developing the theory of equivalence checking (**EC**) and logic synthesis of circuits with a common specification (**CS**) started in [6][4]. A CS S of 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GLSVLSI'05, April 17–19, 2005, Chicago, Illinois, USA.
Copyright ACM 1-59593-057-4/05/0004...$5.00.

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 of 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 circuits N_1 and N_2 implement the same 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 paper 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 function that takes up to 2^k values. 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 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 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. Fortunately, in [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 is given in [4].

The flaw of the procedure of [4] 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 paper, ***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. This important result allows one to specify a specification of a circuit *implicitly* as a partitioning into subcircuits, without representing explicitly the functionality of multi-valued blocks or binary encodings. ***We modify*** the procedure of [4] so that now it not only checks if N_1 and N_2 are functionally equivalent but also tests if specifications of N_1 and N_2 are identical.

The notion of toggle equivalence is key in building a circuit N_2 that implements the same specification S as N_1. In this paper ***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 paper 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. Finally, we draw some conclusions in Section 8.

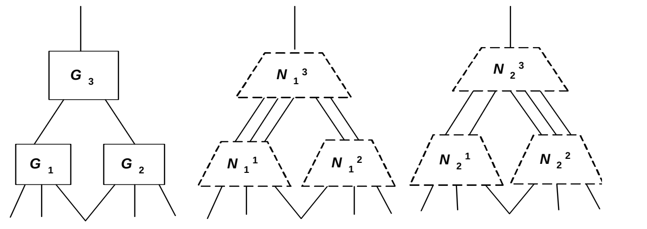

Figure 1. Circuits N_1 and N_2 with a common specification of three blocks

2. TOGGLE EQUIVALENCE

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 Boolean 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 $\boldsymbol{f_1}$=~$\boldsymbol{f_2}$.

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 of $Part(\boldsymbol{f_1})$.

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 Boolean 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 an $|A^X|$ – 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 Proposition 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 Proposition 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.

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 $D_{\text{inp}}(X,Y)$ 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 $D_{\text{out}}(N_1, N_2)$ 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 $D_{\text{inp}}(X,Y)$ be a Boolean function. Denote by $\boldsymbol{D_{\text{out}}(N_1, N_2)}$ the Boolean function obtained from the Boolean function H, where H=$Sat(v(N_1)) \wedge Sat(v(N_2)) \wedge D_{\text{inp}}(X,Y)$, 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 $D_{\text{inp}}(X,Y)$ be a Boolean function relating X and Y. Let $D_{\text{inp}}(X,Y)$ be a correlation function. Then N_1 and N_2 are toggle equivalent under constraint function $D_{\text{inp}}(X,Y)$ iff the function $D_{\text{out}}(N_1,N_2)$ specified in Definition 8 is also a correlation function.

Proof. Only If part. Let N_1 and N_2 be toggle equivalent. Then $D_{\text{out}}(N_1,N_2)$ 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 $D_{\text{out}}(\boldsymbol{z},\boldsymbol{h})=D_{\text{out}}(\boldsymbol{z'},\boldsymbol{h})=1$. 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) $D_{\text{inp}}(\boldsymbol{x}, \boldsymbol{y})=1$ and $D_{\text{inp}}(\boldsymbol{x'}, \boldsymbol{y'})=1$; 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 because 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.

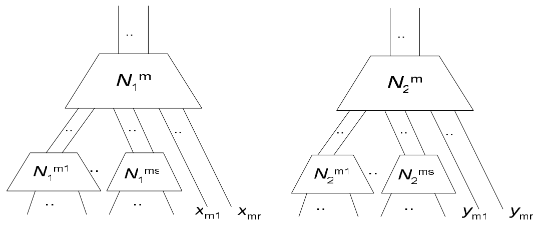

Figure 2. Illustration to Definition 12

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^*.

Definition 10. 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 G_i to G_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)

Definition 11. 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).

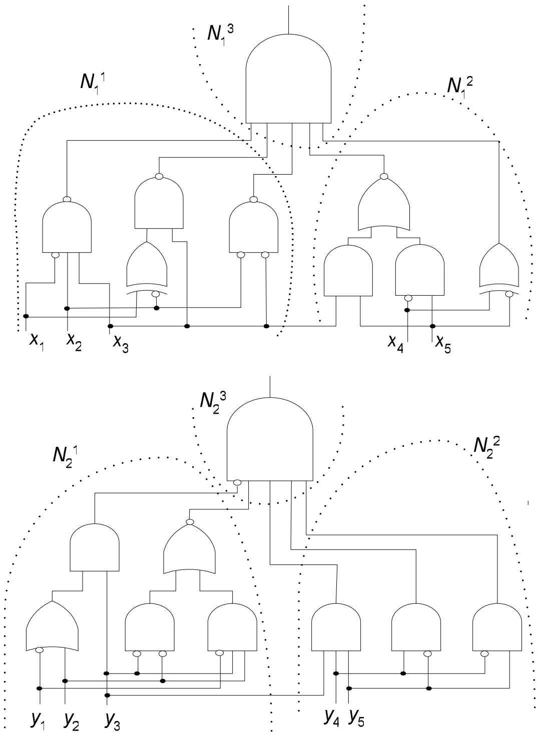

Figure 3. Example of circuits N_1 and N_2 with a CS

Definition 12. 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 $\boldsymbol{D_{out}(N_1^m,N_2^m)}$**, m=1,…, 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_{mt} 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.

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. 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 12.

4. COMMON SPECIFICATION VERIFICATION PROCEDURE

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 [4] 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 10). 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.

```
/* --- $Spec(N_1)= \{N_1^1,..,N_1^k\}$,$Spec(N_2)= \{N_2^1,..,N_2^k\}$ ---*/
$CSV(N_1, N_2, Spec(N_1),Spec(N_2))$ {
 if ($topol\_partition(N_1,N_2,Spec(N_1),Spec(N_2))$ == 'no')
      return('CS_check_failure');

  if ($equiv\_commun\_specs(N_1,N_2,Spec(N_1),Spec(N_2))$ == 'no')
        return('CS_check_failure');

 for ($i$=1; $i$ <= $k$ ; $i$++) {
    $D_{inp}$ = $constr\_func(N_1^i,N_2^i,N_1,N_2)$;
    $D_{out}(N_1^i, N_2^i)$ = $exist\_quantify(N_1^i,N_2^i, D_{inp})$;
    if ($correlation\_function(D_{out})$ == 'no')
         return('CS_check_failure');}

 if ($D_{out}(N_1^k, N_2^k)$ implies $equivalence\_function$)
        return('equivalent');
  else
        return('CS_check_failure');}
```

Figure 4. Pseudocode of the CSV procedure

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 12). 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 13. Let N_1, N_2 be two functionally equivalent circuits with a CS S represented by partitions $Spec(N_1)=\{N_1^1,..,N_1^k\}$, and $Spec(N_2)= \{N_2^1,..,N_2^k\}$. The ***granularity*** of S is the size (i.e. the number of gates) of the largest subcircuit N_i^j, i=1,2, j=1,..,k.

The CSV procedure is exponential in the granularity p of S and 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 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] 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 paper 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][7][8]). 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.

6. ON LOGIC SYNTHESIS 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. (Describing an implementation of this procedure is beyond the scope of this paper. Some encouraging preliminary results are reported in [5].) 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 p of the specification S of N_1 is, the smaller the search space is and the greater the efficiency of the SP procedure is. So, if no good alternative implementation N_2 is found for the current specification of N_1, one can merge some adjacent subcircuits of $Spec(N_1)$ to get a specification of larger granularity. 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 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.

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**

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 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.

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 a 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. 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 specifications 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.

8. CONCLUSIONS

In this paper, we show that two combinational circuits N_1, N_2 have a CS S iff they can be partitioned into toggle equivalent subcircuits. 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. Finally, we give experimental evidence that EC of circuits with unknown CS is hard.

REFERENCES

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

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

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

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

[5] Goldberg, E. *Equivalence checking and logic synthesis of circuits with a common specification.* Technical report. CDNL-TR-2004-1220, August 2004.

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

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

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