# On Complexity of Internal and External Equivalence Checking

Eugene Goldberg, Kanupriya Gulati
*Cadence Design Systems, Texas A&M University,*
*egold@cadence.com, kanu.gulati@gmail.com*

## Abstract

*We compare the complexity of "internal" and "external" equivalence checking. The former is meant for proving the correctness of a synthesis transformation by which circuit $N_2$ is obtained from circuit $N_1$. The latter is meant for proving that circuits $N_1$ and $N_2$ are functionally equivalent without making any explicit assumptions about the origin of $N_1$ and $N_2$. We describe logic synthesis procedures that can produce a circuit $N_2$ whose equivalence with the original circuit $N_1$, most likely, can not be efficiently proved by an external equivalence checker. On the other hand, there are internal equivalence checking procedures that easily prove that $N_1$ and $N_2$ are equivalent. We give experimental data showing that these logic synthesis procedures are not a mathematical curiosity but indeed can be used as a powerful method of logic optimization.*

## 1. INTRODUCTION

Equivalence checking (**EC**) has become an important part of design verification [7]. This success can be attributed to a good scalability of the state-of-the-art equivalence checkers. In turn, this scalability is due to two factors. *First*, logic synthesis tools usually do not re-encode state variables and so EC of two sequential circuits reduces to EC of combinational circuits bounded by registers and/or primary inputs and outputs. (For this reason, in this paper, we consider only EC of combinational circuits and when we refer to a "circuit" we mean a "combinational circuit".) *Second*, in many cases circuits can be represented by small BDDs [4]. Then EC of the corresponding combinational circuits of two designs to be compared can be performed efficiently.

### 1.1 EC of large circuits

The existence of large combinational circuits poses a problem in EC. Informally, we consider a circuit as "large", if its BDD cannot be built efficiently. Typically, this happens when a circuit has a large width [1]. The width of a circuit $N$ describes the amount of communication between different parts of $N$, multiplier being a classical example of a "wide" circuit.

A lot of research in EC has been focused on handling large combinational circuits. In [9], the idea of combining SAT and BDD based methods was explored. The case when circuits to be checked for equivalence have compact BDDs under different variable orders was studied in [13]. The most popular method of handling large circuits is based on employing cut points [2][3]. The idea is to prove functional equivalence of circuits $N_1$ and $N_2$ inductively. First equivalence of some subcircuits of $N_1$ and $N_2$ is established. The outputs of equivalent subcircuits are considered as cut points and new subcircuits are tested for equivalence, inputs of these subcircuits being cut points. This goes on until equivalence of the outputs of $N_1$ and $N_2$ is proven in terms of some cut points. This idea was further developed in [10] and successfully used in many equivalence checkers (e.g. [5][6]). It may happen that even though $N_1$ and $N_2$ are equivalent, they appear to be inequivalent in terms of the chosen cut points. This situation is usually called a false negative. The problem of false negatives was addressed in [12][14].

### 1.2 Internal and external EC

All the methods described above are meant for "**external**" EC. Informally, EC is external if no explicit assumptions are made about the origin of combinational circuits $N_1$ and $N_2$ to be compared. We will say that EC is "**internal**" if it is meant for verification of a logic synthesis transformation by which circuit $N_2$ is obtained from $N_1$. So, by definition, internal EC "knows" the relation between $N_1$ and $N_2$. For simple transformations, internal EC is performed implicitly by using some informal reasoning. Suppose, for example, that internal points $w$ and $w'$ of a circuit $N_1$ are functionally equivalent. Then one can optimize $N_1$ by removing $w'$ and feeding its fan-out nodes with $w$ (instead of $w'$). No explicit procedure is run for EC of $N_1$ and $N_2$ because in this particular case the equivalence of $N_1$ and $N_2$ is "obvious".

Development of new "non-trivial" methods for internal EC is extremely important because these methods enable new logic synthesis procedures. A powerful method of internal EC was introduced in [8] where logic synthesis and EC of circuits with a **common specification** were considered. A specification of a circuit $N$ is just a partition of $N$ into subcircuits. Two circuits $N_1$ and $N_2$ have a common specification if they can be partitioned into $k$ subcircuits $N_1^1,..,N_1^k$ and $N_2^1,..,N_2^k$ such that these subcircuits are connected in "the same way" in $N_1$ and $N_2$ and corresponding subcircuits $N_1^i, N_2^i$ are toggle equivalent. We will refer to the EC procedure of [8] as **EC_TE** where TE stands for toggle equivalence.

The importance of EC_TE is twofold. *First*, it enables a powerful logic synthesis procedure. (We will refer to this procedure as **LS_TE** where LS stands for logic synthesis and TE for toggle equivalence). Given a circuit $N_1$ with specification $N_1^1,..,N_2^k$, LS_TE builds an optimized circuit $N_2$ that is functionally equivalent to $N_1$ by replacing subcircuits $N_1^i$ with their toggle equivalent counterparts in topological order. (The equivalence of $N_1$ and $N_2$ can be established by EC_TE). The power of LS_TE is in its good scalability and flexibility. LS_TE has linear complexity in the number $k$ of subcircuits $N_1^i$ and exponential in the **granularity of specification** which is the size of the largest subcircuit $N_1^i$, in the number of gates. (EC_TE has the same complexity as LS_TE). If a subcircuit $N_1^i$ of $N_1$ has $m$ outputs, then the number of $m$-output subcircuits that are toggle equivalent to $N_1^i$ is huge even for small $m$. So LS_TE enjoys great flexibility even for specifications of very small granularity.

The *second* reason why introduction of EC_TE is important is as follows. Usually, external EC is assumed to be as powerful as internal. The results of [8] imply, that, most probably, there is an exponential gap between external and internal EC. The reason is that finding a common specification of $N_1$ and $N_2$ most likely can not be done efficiently. So EC of $N_1$ and $N_2$ becomes infeasible if the partitioning of $N_1$ and $N_2$ into subcircuits $N_1^1,..,N_1^k$ and $N_2^1,\dots,N_2^k$ is not known. In other words, if $N_1$ is a circuit of large width, external EC of $N_1$ and $N_2$ obtained from $N_1$ by a logic optimization procedure is possible only if this procedure is very "weak". Then $N_1$ and $N_2$ have so much similarity (like existence of many functionally equivalent internal points) that equivalence of $N_1$ and $N_2$ can be proven without any additional information.

### 1.3 Our contribution and structure of the paper

In this paper, we further develop the ideas of [8]. Our contribution is threefold. In [8], no concrete procedure that, given a subcircuit $N_1^i$, generates a toggle equivalent counterpart $N_2^i$, was introduced. So one could consider the performance gap between internal and external EC as a mathematical curiosity. The *first* contribution of this paper is that we give experimental data showing the promise of LS_TE, which makes the theory of [8] much more tangible.

The *second* contribution is that we show that after a slight modification, EC_TE enables a logic synthesis procedure much more powerful than LS_TE. Given a circuit $N_1$ and specification $N_1^1,..,N_1^k$, this procedure is to replace each subcircuit $N_1^i$ with a subcircuit $N_2^i$ that implies toggling of $N_1^i$. We will refer to this procedure as **LS_TI** (where TI stands for toggle implication). LS_TI offers much more flexibility than LS_TE while its complexity is the same as the complexity of LS_TE. Showing that LS_TI "widens" the gap between internal and external EC is our *third* contribution.

This paper is structured as follows. In Section 2 we give basic notions. Section 3 recalls EC_TE and LS_TE procedures of [8]. In Section 4 we describe an EC procedure enabling a logic synthesis procedure LS_TI more powerful than LS_TE. Section 5 gives experimental data showing the power of LS_TE and explains why LS_TI is more powerful than LS_TE. In Section 6 we discuss the complexity of external EC of circuits produced by LS_TE and LS_TI. We conclude with Section 7.

## 2. BASIC NOTIONS

### 2.1 Toggle equivalence of Boolean functions

In this subsection, we recall the notion of toggle equivalence and its properties. All the propositions given in this section are either proven in [8] or can be easily derived from proofs given there.

**Definition 1.** Let $\boldsymbol{f}:\{0,1\}^n \rightarrow \{0,1\}^m$ be an $m$-output Boolean function. A **toggle** of $\boldsymbol{f}$ is a pair of two different output vectors produced by $\boldsymbol{f}$ for two input vectors. In other words, if $\boldsymbol{y}=\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{y}'=\boldsymbol{f}(\boldsymbol{x}')$ and $\boldsymbol{y} \neq \boldsymbol{y}'$, then $(\boldsymbol{y}, \boldsymbol{y}')$ is a toggle.

**Definition 2.** Let $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ 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***.

**Proposition 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. Let $\boldsymbol{f_1}$ be $\boldsymbol{f_2}$ are toggle equivalent. Then there is an invertible function $K$ such that $\boldsymbol{f_1}(\boldsymbol{x})=K(\boldsymbol{f_2}(\boldsymbol{x}))$ and $f_2(\boldsymbol{x})=K^{-1}(\boldsymbol{f_1}(\boldsymbol{x}))$.

Proposition 1 means that if functions $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are toggle equivalent, then there is a one-to-one mapping $K$ between the output vectors produced by $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$.

**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 $\boldsymbol{f_1}=\sim\boldsymbol{f_2}$ where '~' means negation.

Let $N_1$ and $N_2$ be toggle equivalent functions. Definition 3, Definition 4 and Proposition 3 below explain how one can *implicitly* find the mapping $K$ relating outputs produced by $N_1$ and $N_2$.

**Definition 3.** Let $f$ be a Boolean function. We will say that function $f^*$ is obtained from $f$ by ***existentially quantifying away*** variable $x_i$ if $f^* = f(\ldots, x_i=0,\ldots) \vee f(\ldots, x_i=1,\ldots)$.

**Definition 4.** Let $N$ be a circuit. Denote by $v(N)$ the set of variables of $N$. Denote by $\boldsymbol{Sat(v(N))}$ the Boolean function such that $Sat(z)$=1 iff the assignment $z$ to $v(N)$ is "possible" i.e consistent. For example, if $N$ consists of just one AND gate $y=x_1 \wedge x_2$, then $Sat(v(N)) = (\sim x_1 \vee \sim x_2 \vee y) \wedge (x_1 \vee \sim y) \wedge (x_2 \vee \sim y)$. For the sake of simplicity we will denote $Sat(v(N))$ as $\boldsymbol{Sat(N)}$.

**Proposition 3.** Let $N_1$ and $N_2$ be toggle equivalent and $Y_1$, $Y_2$ be the sets of their output variables. Let function $K^*(Y_1,Y_2)$ be obtained from $Sat(N_1) \wedge Sat(N_2)$ by existentially quantifying away the variables of $N_1$ and $N_2$ except those of $Y_1 \cup Y_2$. The function $K^*(Y_1, Y_2)$ implicitly specifies the one-to-one mapping $K$ between output vectors produced by $N_1$ and $N_2$. Namely $K^*(\boldsymbol{y_1}, \boldsymbol{y_2})$ is equal to 1 iff $\boldsymbol{y_1}=K(\boldsymbol{y_2})$.

## 2.2 Implication of toggling

In this subsection, we introduce the notion of implication of toggling.

**Definition 5.** Let $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ be two multi-output functions with the same set of variables $X=\{x_1,\ldots,x_n\}$. Toggling of function $\boldsymbol{f_1}$ **implies toggling of** $\boldsymbol{f_2}$ (denoted as $\boldsymbol{f_1} \leq \boldsymbol{f_2}$), if for any pair of assignments $\boldsymbol{x}'$, $\boldsymbol{x}''$ to $X$, $\boldsymbol{f_1}(\boldsymbol{x}') \neq \boldsymbol{f_1}(\boldsymbol{x}'')$ implies $\boldsymbol{f_2}(\boldsymbol{x}') \neq \boldsymbol{f_2}(\boldsymbol{x}'')$.

**Definition 6.** Toggling of a multi-output function $\boldsymbol{f_1}(\boldsymbol{x_1},..,\boldsymbol{x_n})$ **strictly implies toggling of** a multi-output function $\boldsymbol{f_2}(\boldsymbol{x_1},..,\boldsymbol{x_n})$ (denoted as $\boldsymbol{f_1} < \boldsymbol{f_2}$) if $\boldsymbol{f_1} \leq \boldsymbol{f_2}$ but $\boldsymbol{f_2} \leq \boldsymbol{f_1}$ does not hold.

**Remark 1.** We will **denote** by $N_1 \leq N_2$ (respectively $N_1 < N_2$) the fact that toggling of the function implemented by Boolean circuit $N_1$ implies toggling of (respectively strictly implies toggling of) the function implemented by Boolean circuit $N_2$.

**Proposition 4.** Boolean functions $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are toggle equivalent iff $\boldsymbol{f_1} \leq \boldsymbol{f_2}$ and $\boldsymbol{f_1} \leq \boldsymbol{f_2}$.

**Proposition 5.** 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. Let $\boldsymbol{f_1} \leq \boldsymbol{f_2}$. Then there is a function $K$ such that $\boldsymbol{f_1}(\boldsymbol{x})=K(\boldsymbol{f_2}(\boldsymbol{x}))$.

Note that unless $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are toggle equivalent, the function $K$ is not invertible.

## 2.3 Testing toggle implication and toggle equivalence

In this subsection, we describe how toggle equivalence and implication of toggling can be tested. Let $N_1$ and $N_2$ be two Boolean circuits to be checked for implication of toggling. Let $X=\{x_1,.., x_n\}$ be the set of input variables of $N_1, N_2$. 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 $N_1 \leq N_2$ holds iff the function $S = H(N_1, N_2) \wedge H(N^*_1, N^*_2) \wedge Neq(Y, Y^*) \wedge Eq(Z, Z^*)$ is unsatisfiable (i.e. it is a constant 0). Here $N^*_1$ and $N^*_2$ are copies of circuits $N_1$ and $N_2$, with input variables $X^*=\{x^*_1,.., x^*_n\}$ and output variables $Y^*=\{y^*_1,\ldots, y^*_m\}$ and $Z^*= \{z^*_1,.., z^*_k\}$ respectively. The function $H(N_1, N_2)$ is equal to $Sat(N_1) \wedge Sat(N_2)$. The value of $Eq(\boldsymbol{y}, \boldsymbol{y^*})$ where $\boldsymbol{y}$ and $\boldsymbol{y^*}$ are assignments to $Y$ and $Y^*$ respectively is equal to 1 iff $\boldsymbol{y}=\boldsymbol{y^*}$. The function $Neq(Y, Y^*)$ is the negation of $Eq(Y, Y^*)$.

Indeed, $S$=1 means that for a pair of input vectors $\boldsymbol{x}$ and $\boldsymbol{x^*}$, circuit $N_1$ toggles (which sets $Neq(Y, Y^*)$ to 1) while $N_2$ does not (which sets $Eq(Z,Z^*)$ to 1).

From Proposition 4 it follows that checking for toggle equivalence reduces to two satisfiability checks (SAT-checks for short).

### 2.4 Correlation function

In this subsection, we use the notion of correlation function to extend definitions of toggle implication and toggle equivalence to the case when functions $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ have different sets of variables.

**Definition 7.** Let $X$ and $X^*$ be two disjoint sets of Boolean variables (the number of variables in $X$ and $X^*$ may be different). A function $D(X,X^*)$ is called a ***correlation function*** if there are subsets $Q^X \subseteq \{0,1\}^{|X|}$ and $Q^{X^*} \subseteq \{0,1\}^{|X^*|}$ such that $D(X,X^*)$ specifies a bijective mapping $M$: $Q^X \rightarrow Q^{X^*}$. Namely $D(\boldsymbol{x}, \boldsymbol{y})=1$ iff $\boldsymbol{x} \in Q^X$ and $\boldsymbol{y} \in Q^{X^*}$ and $\boldsymbol{y} = M(\boldsymbol{x})$.

Let $\boldsymbol{f_1}(X)$ and $\boldsymbol{f_2}(X^*)$ be two multi-output Boolean functions where $X=\{x_1,\ldots, x_k\}$ and $X^* =\{x^*_1,\ldots, x^*_p\}$ are sets of their variables. (Note, that $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ may have different number of variables.). Let $D(X, X^*)$ be a correlation function relating variables of $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$. Then one can introduce notions of toggle equivalence and toggle implication for $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$. The only difference from definitions and results listed in subsections 2.1,Proposition 3,2.3 is that now one should consider only assignments that satisfy $D(X,X^*)$.

Let us show how this works for toggle equivalence. Functions $\boldsymbol{f_1}(X)$ and $\boldsymbol{f_2}(X^*)$ are said to be toggle equivalent under input constraint $D(X,X^*)$, if for any two pairs $(\boldsymbol{x}, \boldsymbol{x^*})$ and $(\boldsymbol{y}, \boldsymbol{y^*})$ of input vectors such that $D(\boldsymbol{x}, \boldsymbol{x^*})=D(\boldsymbol{y}, \boldsymbol{y^*})=1$, it is true that $\boldsymbol{f_1}(\boldsymbol{x}) \neq \boldsymbol{f_1}(\boldsymbol{y}) \Leftrightarrow \boldsymbol{f_2}(\boldsymbol{x^*}) \neq \boldsymbol{f_2}(\boldsymbol{y^*})$. The mapping between output vectors produced by toggle equivalent circuits $N_1$ and $N_2$ (implementing functions $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ respectively), can be obtained from $Sat(N_1) \wedge Sat(N_2) \wedge D(X, X^*)$ by existentially quantifying away all the variables of $N_1$ and $N_2$ except output variables. The other results and definitions of subsections 2.1,Proposition 3,2.3 can be modified in a similar manner.

## 3. LOGIC SYNTHESIS AND EC OF CIRCUITS WITH COMMON SPECIFICATION

In this section, we recall LS_TE and EC_TE procedures of [8]. From now on we assume that circuit $N_1$ to be optimized has only one output. (If a circuit to be optimized has more than one output, then the LS_TE procedure can be separately applied to every subcircuit feeding an output of $N_1$).

### 3.1 Logic synthesis preserving toggle equivalence

In this subsection, we recall the procedure of Logic Synthesis preserving Toggle Equivalence (abbreviated as **LS_TE**) introduced in [8]. The pseudocode of the LS_TE procedure is shown in Figure 1.

1 *LS_TE*($N_1$, *Spec*($N_1$),*cost_function*) {
2 for ($i$=1; $i$ <= $k$ ; $i$++) {
3 $D_{inp}(N_1^i, N_2^i)$= *constraint_function*($N_1$,$N_2$,$i$);
4 $N_2^i$ = *synth_toggle_equivalent*($N_1^i$, $D_{inp}$,*cost_function*)
5 $D_{out}(N_1^i, N_2^i)$ = *exist_quantify*($N_1^i$,$N_2^i$, $D_{inp}$); }
6 return($N_2$,*Spec*($N_2$))}

**Figure 1. Pseudocode of the LS_TE procedure**

Following [8] we also assume that specification $Spec(N_1) = \{N_1^1,.., N_1^k\}$ (i.e. the initial partition of circuit $N_1$ into subcircuits $N_1^i$) is topological. Let $G$ be a directed graph whose nodes are subcircuits $N_1^i$ and an edge of $G$ directed from node $N_1^i$ to node $N_1^j$ implies that an output of $N_1^i$ is connected to an input of $N_1^j$. We will call $G$ a **specification graph**. $Spec(N_1)$ is a **topological specification** if its specification graph $G$ is acyclic.

Given a circuit $N_1$ with specification $Spec(N_1) = \{N_1^1,.., N_1^k\}$, LS_TE builds circuit $N_2$ with specification $Spec(N_2)= \{N_2^1,.., N_2^k\}$ that is identical to $Spec(N_1)$. Let $G_1$ and $G_2$ be directed graphs describing connections of subcircuits $N_1$ and $N_2$ as described above. $Spec(N_1)$ and $Spec(N_2)$ are considered to be **identical specifications** if a) $G_1$ and $G_2$ are acyclic; b) $G_1$ and $G_2$ are isomorphic; c) subcircuits $N_1^i$ and $N_2^i$ are toggle equivalent, $i$=1,..,$k$ in terms of their inputs related by a constraint function that is a correlation function (see below).

*Since* $Spec(N_1)$ is topological, one can assign levels to subcircuits $N_1^i$. We assume that subcircuits $N_1^i$ are numbered "topologically" i.e. if $i < j$ then $topol\_level(N_1^i) \leq topol\_level(N_1^j)$. The LS_TE procedure builds circuit $N_2$ by replacing subcircuits $N_1^i$, $i$=1,..,$k$ with their toggle equivalent counterparts $N_2^i$ in topological order, from inputs to outputs.

Let us consider how LS_TE works by the example of Figure 2. LS_TE starts with subcircuit $N_1^1$ and recovers the function $D_{inp}(N_1^1, N_2^1)$ relating inputs of $N_1^1$ and subcircuit $N_1^2$ to be built (line 3 of the pseudocode). The inputs of $N_1^1$ are inputs of $N_1$ (and so $N_1^1$ has the lowest topological level 1). In that case, $D_{inp}(N_1^1, N_2^1)$ is just a conjunction of equality functions relating corresponding inputs of $N_1^1$ and $N_2^1$ (and so $D_{inp}(N_1^1, N_2^1)$ is a correlation function "identifying" the corresponding inputs of $N_1^1$ and $N_2^1$.) Then an actual subcircuit $N_2^1$ toggle equivalent to $N_1^1$ is synthesized (line 4). In the end of this iteration, the function $D_{\text{out}}(N_1^1, N_2^1)$ relating outputs of $N_1^1$ and $N_2^1$ is built (line 5) as described in Proposition 3. Since $N_1^1$ and $N_2^1$ are toggle equivalent, there is a one-to-one mapping between the output vectors they produce. So $D_{\text{out}}(N_1^1, N_2^1)$ is a correlation function.

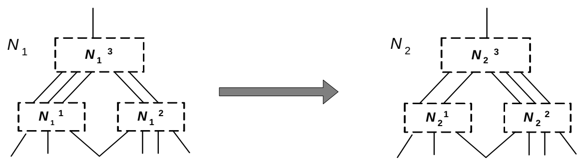


**Figure 2. Optimization of $N_1$ by LS_TE**

Then, the LS_TE procedure processes subcircuit $N_1^2$ in the same manner, generating a toggle equivalent subcircuit $N_2^2$ and the correlation function $D_{\text{out}}(N_1^2, N_2^2)$. Finally, the subcircuit $N_1^3$ is processed similarly to $N_1^1$ and $N_1^2$ with one exception. The inputs of $N_1^3$ are fed by the outputs of $N_1^1$ and $N_1^2$. So now the function $D_{\text{inp}}(N_1^3, N_2^3)$ relating inputs of $N_1^3$ and $N_2^3$ (synthesized in line 4) equals $D_{\text{out}}(N_1^1, N_2^1) \wedge D_{\text{out}}(N_1^2, N_2^2)$. (To obtain $D_{\text{inp}}(N_1^3, N_2^3)$ one has to take the conjunction of $D_{\text{out}}(N_1^i, N_2^i)$ for all the subcircuits whose outputs feed inputs of $N_1^3$ and $N_2^3$. In this particular case, these subcircuits are $N_1^1, N_1^2$ and $N_2^1, N_2^2$.) It is not hard to show that a conjunction of correlation functions is a correlation function too and so $D_{\text{inp}}(N_1^3, N_2^3)$ is a correlation function.

Let us $N_2^3$ be a circuit toggle equivalent to $N_1^3$ under constraints $D_{\text{inp}}(N_1^3, N_2^3)$. Recall that $N_1^3$ has only one output. It is not hard to show that then $N_2^3$ either has one output or all outputs of $N_2^3$ but one can be removed without affecting its being toggle equivalent to $N_1^3$. (So we will assume that $N_2^3$ has one output.) Since $N_1^3$ and $N_2^3$ are single-output subcircuits, their toggle equivalence, means that they are functionally equivalent modulo negation. So the circuit $N_2$ consisting of subcircuits $N_2^1, N_2^2, N_2^3$ is functionally equivalent to $N_1$ (modulo negation).

## 3.2 EC of circuits with a common specification

The EC of $N_1$ and a circuit $N_2$ obtained from $N_1$ by LS_TE can be done by the **EC_TE** procedure of [8] whose (slightly changed) pseudocode is shown in Figure 3. EC stands for equivalence checking and TE stands for toggle equivalence. Recall that $N_1$ is a single-output circuit and, as we mentioned above, LS_TE builds a circuit $N_2$ that has one output too.

The input to the EC_TE procedure are circuits $N_1$ and $N_2$ and their specifications $Spec(N_1)=\{N_1^1,.., N_1^k\}$, $Spec(N_2)= \{N_2^1,.., N_2^k\}$. So EC_TE is an internal EC procedure. EC_TE is incomplete in the sense that it gives a definite answer only if $Spec(N_1)$ and $Spec(N_2)$ are identical. Otherwise, EC_TE returns the answer '*CS_check_failure*', which means that specifications of $N_1$ and $N_2$ are different.

**1** *EC_TE*($N_1$, $N_2$, *Spec*($N_1$),*Spec*($N_2$)) {
**2** if (*topol_spec*($N_1$, *Spec*($N_1$)) == '*no*') || *topol_spec*($N_2$, *Spec*($N_2$)) == '*no*'))
**3** return('*CS_check_failure*');
**4** if (*graph_isomorphism* ( $N_1$, $N_2$, *Spec*($N_1$),*Spec*($N_2$)) == '*no*')
**5** return('*CS_check_failure*');
**6** for ($i$=1; $i$ <= $k$ ; $i$++) {
**7** $D_{inp}(N_1^i, N_2^i)$= *constraint_function*($N_1$,$N_2$,$i$);
**8** if (*toggle_equiv*(($N_1^i$, $N_2^i$, $D_{inp}$)) == '*no*') return('*CS_check_failure*');
**9** $D_{out}(N_1^i, N_2^i)$ = *exist_quantify*($N_1^i$,$N_2^i$, $D_{inp}$);}
**10** if ($D_{out}(N_1^k, N_2^k)$) is '*equivalence_function*') return('*equivalent*');
**11** else return('*inequivalent*');}

**Figure 3. Pseudocode of the EC_TE procedure**

Let $G_1$, $G_2$ be specification graphs for *Spec*($N_1$) and *Spec*($N_2$) respectively (see subsection 3.1.) In line 2, EC_TE checks if graphs $G_1$ and $G_2$ are acyclic. In line 4, EC_TE checks if $G_1$ and $G_2$ are isomorphic. If either check fails, '*CS_check_failure*' is reported.

The main work is done in the 'for' loop (lines 6-9) where subcircuits $N_1^i$ and $N_2^i$ ($i$=1,..,$k$) are checked for toggle equivalence in topological order. First, in line 7 the function $D_{inp}(N_1^i, N_2^i)$ relating inputs of $N_1^i$ and $N_2^i$ is formed exactly as it is done by LS_TE (Figure 1, line 3). In line 8, EC_TE checks if $N_1^i$ and $N_2^i$ (whose inputs are related by the function $D_{\text{inp}}(N_1^i, N_2^i)$ computed in line 7) are toggle equivalent. If not, then EC_TE returns '*CS_check_failure*'. (As we showed in subsection 2.3, checking of toggle equivalence reduces to two SAT-checks.) In line 9, the function $D_{out}(N_1^i, N_2^i)$ relating outputs of $N_1^i$ and $N_2^i$ is computed. This is done by existentially quantifying away from $Sat(N_1^i) \wedge Sat(N_2^i) \wedge D_{inp}(N_1^i, N_2^i)$ all the variables except the output variables of $N_1^i$ and $N_2^i$.

After finishing the 'for' loop, in lines 10-11, EC_TE checks if the function $D_{out}(N_1^k, N_2^k)$ (note that the outputs of $N_1^k$ and $N_2^k$ are outputs of $N_1$ and $N_2$ respectively) is an equivalence function. If it is, then $N_1$ and $N_2$ are equivalent. Otherwise, $N_1$ and $N_2$ are complements of each other (because $N_1^k$ and $N_2^k$ are toggle equivalent but not functionally equivalent) and so they are inequivalent.

The complexity of both EC_TE and LS_TE is linear in the number of subcircuits in $N_1$ and $N_2$ and exponential in granularities of $Spec(N_1)$ and $Spec(N_2)$. (Recall that **granularity** of $Spec(N_1)$ is the number of gates in the largest subcircuit $N_1^i$, $i$=1,..,$k$.)

# 4. NEW EC AND LOGIC SYNTHESIS PROCEDURES

In this section, we describe an extension of the EC_TE and LS_TE procedures called **EC_TI** and **LS_TI** respectively (where TI stands for toggle implication). First, we give a generic method for introducing a logic transformation through an "enabling" procedure and explain how this method works for EC_TE and LS_TE. Then we introduce EC_TI and LS_TI.

## 4.1 A generic method for introducing logic transformation

The idea of the method is to introduce a logic transformation through an Enabling internal EC procedure (we will refer to it a **EEC** ). The input to an EEC is an original circuit $N_1$, a modified circuit $N_2$ and some information about the transformation $T$ used to obtain $N_2$ from $N_1$. EEC has to be sound. That is, if EEC says that $N_1$ and $N_2$ are equivalent (or inequivalent) it has to give the right answer. Besides, EEC should be able to recognize if $N_2$ can not be obtained from $N_1$ by the transformation $T$. After designing an EEC, one formulates a logic synthesis procedure that given a circuit $N_1$ generates a circuit $N_2$ whose equivalence to $N_1$ can be verified by this EEC. We will say that this synthesis procedure is enabled by this EEC.

Let us illustrate how this method works for introducing LS_TE. The EC_TE procedure satisfies the requirements above for an EEC. Indeed, the input to EC_TE consists of circuits $N_1$ and $N_2$ and partitions $Spec(N_1)$, $Spec(N_2)$ as information about the synthesis transformation to be enabled. The soundness of EC_TE trivially follows from the fact that EC_TE returns the 'equivalent' (or 'inequivalent') answer only if it correctly derived the equivalence (respectively inequivalence) function relating the outputs of $N_1$ and $N_2$. It is not hard to see that LS_TE is exactly the procedure enabled by EC_TE. Indeed, EC_TE checks in topological order if subcircuits $N_1^i$ and $N_2^i$ of $Spec(N_1)$ and $Spec(N_2)$ are toggle equivalent. In turn, LS_TE builds $N_2$ by generating toggle equivalent counterparts of subcircuits $N_1^i$ in topological order.

## 4.2 Introduction of LS_TI

Let EC_TI be an EC procedure that is different from EC_TE only in one aspect. Instead of checking if $N_1^i$ and $N_2^i$ (line 8 of Figure 3) are toggle equivalent it checks if toggling of $N_1^i$ implies that of $N_2^i$ . That is instead of checking if both $N_1^i \leq N_2^i$ and $N_2^i \leq N_1^i$ hold it just checks if $N_1^i \leq N_2^i$. The EC_TI procedure is sound for the same reason as EC_TE. (That is it correctly derives an (in)equivalence function relating the outputs of $N_1$ and $N_2$. The correctness of derivation follows from the "soundness" of existential quantification that is used to obtain functions $D_{out}(N_1^i, N_2^i)$ i=1,..,$k$)

In the context of the EC_TI procedure, we will say that circuits $N_1$ and $N_2$ have **identical specifications** $Spec(N_1)$= $\{N_1^1,.., N_1^k\}$ and $Spec(N_2)=\{N_2^1,..,N_2^k\}$ if specification graphs $G_1$ and $G_2$ are acyclic, isomorphic and $N_1^i \leq N_2^i$ , $i$=1,..,$k$-1 and for the last value of $i$ (i.e. $i$=$k$), subcircuits $N_1^k$ and $N_2^k$ are toggle equivalent. The EC_TI procedure either correctly identifies circuits $N_1$ and $N_2$ as (in)equivalent or reports that $Spec(N_1)$ and $Spec(N_2)$ are not identical.

Now we define a procedure enabled by EC_TE. Let $N_1$ be a circuit and $Spec(N_1)=\{N_1^1,\ldots,N_1^k\}$ be its specification. Let LS_TI be a logic synthesis procedure that is different from LS_TE only in two aspects. *First*, for $i$=1,..,$k$-1 LS_TI generates subcircuit $N_2^i$ such that toggling of $N_1^i$ implies that of $N_2^i$ i.e. $N_1^i \leq N_2^i$ (So, LS_TI is different from LS_TE in line 4 of Figure 1). Only for $i$=$k$, LS_TI generates subcircuit $N_2^i$ that is toggle equivalent to $N_1^i$.

*Second*, when generating a subcircuit $N_2^i$, LS_TI limits the number of outputs of $N_2^i$. The reason is as follows. Let $D_{inp}(N_1^i, N_2^i)$ be the constraint function relating inputs of $N_1^i$ and the subcircuit $N_2^i$ to be built. If the number of outputs in $N_2^i$ is not limited, then the simplest subcircuit $N_2^i$ such that $N_1^i \leq N_2^i$ is the **identity circuit $\boldsymbol{I_p}$** where $p$ is the number of inputs in $N_2^i$. ($I_p$ is a circuit with inputs $x_1$,..,$x_p$ and $p$ outputs $y_1$,..$y_p$, implementing the functions $y_i$=$x_i$, $i$=1,.., $p$. So $I_p$ does not have any gates.) Hence, without limiting the number of outputs in $N_2^i$, LS_TI would just generate identity circuits as $N_2^i$, $i$=1,..,$k$-1 pumping all the functionality into functions $D_{inp}(N_1^i,N_2^i)$. Only when generating the subcircuit $N_2^k$ (which is toggle equivalent to $N_1^k$ and has one output) a non-empty subcircuit would be generated. It can be shown that in this case $D_{inp}(N_1^k, N_2^k)$, would essentially describe the relation between inputs of $N_1^k$ and the primary inputs of $N_1$. This means that if one does not limit the number of outputs in $N_2^i$, the LS_TI procedure would delay all the synthesis work until $i=k$ and so it would not be scalable. The simplest way to limit the number of outputs in $N_2^i$ is to require that $num\_of\_outputs(N_2^i) \leq num\_of\_outputs(N_1^i)$. Under such a restriction, LS_TI has the same complexity as LS_TE i.e. it is linear in the number of subcircuits in $Spec(N_1)$ and exponential in the granularity of $Spec(N_1)$.)

LS_TI is a synthesis procedure enabled by EC_TI. That is, if for a given $N_1$ and specification $Spec(N_1)$, LS_TI builds a circuit $N_2$ with specification $Spec(N_2)$, EC_TI will prove $N_1$ and $N_2$ to be equivalent.

## 5. BIG PROMISE OF LOGIC SYNTHESIS PRESERVING COMMON SPECIFICATION

In this section we give some experimental data showing the power of the LS_TE procedure and discuss the potential of LS_TI that should be much more powerful than LS_TE.

### 5.1 Power of LS_TE

The key procedure of LS_TE is called in the loop (line 4 of Figure 1) $k$ times to generate a subcircuit $N_2^i$ that is toggle equivalent to $N_1^i, i=1,..,k$. We will refer to it as the **TEP** procedure (TEP stands for **Toggle Equivalence Preserving**). Such a procedure has been developed in [17].

Let $M'$ and $M''$ denote subcircuits $N_1^i$ and $N_2^i$ respectively. Given $M'$ and a constraint (correlation function) imposed on inputs $M'$ and $M''$, the TEP procedure of [17] builds $M''$ as follows. It constructs a sequence of circuits $M_1, M_2,..$, such that $M \leq M_{i+1} < M_i$. (Here $M_1$ is an identity circuit $I_p$ and $p$ is the number of inputs in $M''$). That is $M_{i+1}$ toggles at least "as much" as $M$, but "strictly less" than $M_i$. Since every circuit $M_{i+1}$ of this sequence loses at least one toggle of $M_i$, this sequence converges to a circuit $M_s$ such that $M_s \leq M$ and $M \leq M_s$. This means that $M_s$ is toggle equivalent to $M'$ and so $M_s$ is the final circuit $M''$. (A more detailed description of the TEP procedure is beyond the scope of this paper. )

In this paper we give some experimental data on *our* implementation of the TEP procedure for optimization of multi-output circuits just to show that LS_TE has a great practical potential. So the fact that external EC of circuits produced by LS_TE is problematic is significant.

**Table 1. Generation of toggle equivalent circuits**

| Circuit | #in-puts | Initial #outputs | TEP #outputs | SIS #gates | TEP #gates |
|---|---|---|---|---|---|
| squar5 | 5 | 8 | 8 | 60 | **4** |
| rd84 | 8 | 4 | 8 | 174 | **52** |
| 5xp1 | 7 | 10 | 7 | 140 | **0** |
| b1 | 3 | 4 | 3 | 11 | **2** |
| bw | 5 | 28 | 8 | 155 | **9** |
| cm138 | 6 | 8 | 4 | 28 | **15** |
| cm42a | 4 | 10 | 6 | 31 | **6** |
| cm82a | 5 | 3 | 5 | 21 | **18** |
| exp5p | 8 | 63 | 19 | 286 | **131** |
| f51m | 8 | 8 | 8 | 101 | **0** |
| con1 | 7 | 4 | 8 | **82** | 94 |
| sqrt | 8 | 4 | 9 | **76** | 90 |

In Table 1 we compare the results of optimization of some MCNC benchmarks by SIS [15] and by the TEP procedure. The name of the circuit and the number of inputs and outputs are shown in the first three columns of Table 1. The results of optimization by SIS with the script 'rugged' followed by technology decomposition (to obtain a circuit of two-input AND gates and invertors) is shown in the fifth column. The results of using TEP to build a toggle equivalent circuit are shown in the fourth (the number of outputs) and sixth (the number of gates) columns.

For the majority of circuits TEP was able to find much smaller toggle equivalent counterparts. In two cases (5xp1 and f51m) , TEP removed all the logic. This means that, for example, for different input assignments, circuit 5xp1 generates different output assignments. So the identity circuit $I_7$ is toggle equivalent to 5xp1.

Of course, such re-encoding of output assignments of the original circuit requires changing the surrounding logic. To explain why re-encoding may still lead to significant logic reduction let us consider the following example. Suppose that a circuit $N_1$ to be optimized consists of two subcircuits, $N_1^1$ and $N_1^2$ where the outputs of $N_1^1$ are connected to inputs of $N_1^2$. Suppose circuits $N_1^1$ and $N_1^2$ were built "independently". That is when designing circuit $N_1^1$, the output encoding for $N_1^1$ was chosen without any consideration of circuit $N_1^2$. Then, in a sense, any circuit toggle equivalent to $N_1^1$ is as good as $N_1^1$. So, it is a reasonable heuristic to build a circuit $N_2^1$ that is the smallest toggle equivalent counterpart of $N_1^1$ and then try to find the smallest subcircuit $N_2^2$ that is toggle equivalent to $N_1^2$ under constraints specified by the function $D_{out}(N_1^1, N_1^2)$.

Table 2 shows results of applying LS_TE with the heuristic above to logic optimization of some arithmetic expressions with an integer variable $x$, $x \geq 0$. (The second column of Table 2 gives the number of bits in $x$. ) Each circuit $N_1$ of Table 2 consists of subcircuits $N_1^1$ and $N_1^2$. First three circuits $N_1$ implement Boolean function $x^2 < C_1$. Here $N_1^1$ implements the function $x^2$ and subcircuit $N_1^2$ implements comparison with a constant $C_1$. The last three circuits $N_1$ implement Boolean function $C_1*x < C_2$. Here, subcircuit $N_1^1$ implements the function $C_1*x$, and $N_1^2$ implements comparison with a constant $C_2$. Circuits $N_1^1$ and $N_1^2$ were built from standard combinational blocks. (So, for example, in our experiments, $N_1^1$ was a derivative of a regular multiplier.)

**Table 2. Optimization by LS_TE**

| Circuit | #bits | $C_1$ | $C_2$ | SIS #gates | LS_TE #gates |
|---|---|---|---|---|---|
| $x^2 < C_1$ | 6 | 200 | - | 196(12) | 5 |
| $x^2 < C_1$ | 7 | 200 | - | 265 (16) | 6 |
| $x^2 < C_1$ | 7 | 500 | - | 273(15) | 6 |
| $C_1*x < C_2$ | 7 | 49 | 300 | 84(15) | 6 |
| $C_1*x < C_2$ | 7 | 111 | 300 | 160(12) | 6 |
| $C_1*x < C_2$ | 7 | 49 | 500 | 56(14) | 6 |

It is not hard to see that $x^2 < C_1$ is equivalent to $x < C'_1$ where $C'_1$ is the constant equal to *ceiling*(*square_root*($C_1$)). Similarly, $C_1*x < C_2$ is equivalent to $x < C'_2$ where $C'_2$=*ceiling*($C_2/C_1$). So there is a very simple circuit implementation of either Boolean function.

The results of optimization by SIS are shown in the fifth column. The first number of this column gives the number of gates in the circuit obtained after applying script 'rugged' and technology decomposition. The number in parenthesis gives the number of gates in the resulting circuit after applying script 'rugged' many times until the solution stabilizes and then running technology decomposition.

The results of applying LS_TE (that used the TEP procedure of [17]) are shown in the last column. Note that $N_1^1$ (for both $x^2$ and $C_1*x$ cases) generates different output assignments for different input assignments. This means that the identity circuit $I_m$ (where $m$ is the number of bits in $x$) is toggle equivalent to $N_1^1$. So LS_TE picked $I_m$ as $N_2^1$. Then it computed the function $D_{out}(N_1^1, N_2^1)$ relating outputs of $N_1^1$ and $N_2^1$ and built subcircuit $N_2^2$ toggle equivalent to $N_1^2$ under constraints specified by $D_{out}(N_1^1, N_2^1)$. The size of circuits obtained by LS_TE is smaller than those of SIS even after multiple applications of the script 'rugged'. (Since a circuit built by LS_TE is functionally equivalent to the original one, it is a fair comparison.)

Note, that indeed LS_TE applied the heuristic mentioned above. Since $N_1^1$ implementing, say, $x^2$ was built without any consideration of $N_1^2$ any circuit toggle equivalent to $N_1^1$ is as "good" as $N_1^1$. So LS_TE picked the simplest such a circuit that is $I_m$.

## 5.2 Potential of LS_TI

LS_TI is more powerful than LS_TE because toggle implication is a more general relation than toggle equivalence. So LS_TI is much more flexible than LS_TE (while its complexity is the same as that of LS_TE).

Let $N_1^i$ be a subcircuit of circuit $N_1$ with specification $Spec(N_1)=\{N_1^1,., N_1^k\}$. If $N_2^i$ is a subcircuit synthesized by LS_TE that is toggle equivalent to $N_1^i$ (under constraints specified by function $D_{inp}(N_1^i, N_2^i)$), every toggle of $N_2^i$ has a "matching" toggle of $N_1^i$. On the other hand, if $N_2^i$ is built by LS_TI, it may have toggles that are not matched by $N_1^i$ (because LS_TI has to preserve only $N_1^i \leq N_2^i$). Since LS_TI builds a circuit $N_2$ that is functionally equivalent to $N_1$, these unmatched toggles of $N_2^i$ do not reach the output of $N_2$. The blocking of the unmatched toggles is done "automatically".

Let us consider advantage of LS_TI over LS_TE by a simple example. Suppose that one needs to implement Boolean function $f(x) < 9$ where $x$ is an $m$-bit integer. Let $f(x)$ be equal to $x^2$ at all the $2^m$ points except for the point $x$=4 where $f(4)$ is equal to 25 (instead of 16). It is not hard to see that the expression $f(x) < 9$ is equivalent to $x < 3$. Suppose that $f(x) < 9$ is implemented by a circuit $N_1$ that is a composition of subcircuits $N_1^1$ and $N_1^2$ where $N_1^1$ implements $f(x)$ and $N_1^2$ implements comparison with 9.

Suppose that the LS_TE procedure is applied to optimize $N_1$. LS_TE can not use $I_m$ as $N_2^1$, because $f(4)=f(5)$= 25 and so $I_m$ is not toggle equivalent to $N_1^1$. So LS_TE would build $N_2^1$ implementing some "non-trivial" function of $x$. Hence, $N_2^2$ would have to implement a more complex function than $x < 3$.

Now suppose that $N_1$ is optimized by LS_TI. Note that toggling of $N_1^1$ implies toggling of the identity circuit $I_m$. So LS_TI can use $I_m$ as subcircuit $N_2^1$. Then LS_TI can build subcircuit $N_2^2$ implementing $x < 3$. So due to greater flexibility LS_TI is able to generate a smaller circuit than LS_TE.

To become practical, LS_TI needs a procedure that, given a subcircuit $N_1^i$ and a constraint function $D_{inp}$, builds an optimized subcircuit $N_2^i$ such that a) $N_1^i \leq N_2^i$ under constraint $D_{inp}$; b) the number of outputs in $N_2^i$ is limited by a constant. (We will refer to this procedure as **Toggle Implication Preserving** or **TIP**.) Interestingly, the TEP procedure of [17] briefly sketched in the previous subsection can be also used as a TIP procedure. Given a circuit $M'$, the TEP procedure above builds a sequence of circuits $M_1$,$M_2$, .. such that $M_1$ is an identity circuit $I_p$ and $M \leq M_{i+1} < M_i$. The TEP procedure stops when $M_s$ is toggle equivalent to $M$. Suppose the TEP procedure stops as soon as the number of outputs in the current circuit $M_i$ is below a threshold. Then $M \leq M_i$ and the number of outputs in $M_i$ is bounded by a specified threshold. So the TEP procedure of [17] with the new termination condition is a TIP procedure.

Let $N_1^i$ be a subcircuit $N_1$ to be re-synthesized by LS_TI. Let $N_1^i$ have $k$ outputs and we want to build a subcircuit $N_2^i$ of $k$ outputs. The number of $k$-output Boolean functions $\boldsymbol{f}_2^i$ such that $\boldsymbol{f}_1^i \leq \boldsymbol{f}_2^i$ ,where $\boldsymbol{f}_1^i$ is the function implemented by $N_1^i$, is no less than $(2^k!)$. Indeed, even if some output assignments of $N_1^i$ are unsatisfiable, there is always subcircuit $N_2^i$ (if its number of inputs of $N_2^i$ is greater or equal to $k$) such that all $2^k$ output assignments of $N_2^i$ are satisfiable and $N_1^i \leq N_2^i$. Let this be the case and $\boldsymbol{f}_2^i$ be the function implemented by $N_2^i$. Any permutation of $2^k$ output assignments in the truth table of $\boldsymbol{f}_2^i$ gives a new function $\boldsymbol{f}'^i_2$ such that $\boldsymbol{f}_1^i \leq \boldsymbol{f}'^i_2$.

The value of $(2^k!)$ is huge even for small $k$ (e.g. if $k$=5, then $(2^5)!$ is equal to $2.6*10^{35}$). So even if $Spec(N_1)$ has a very small granularity, LS_TI still enjoys great flexibility. On the other hand, since the complexity of LS_TI is linear in the number of subcircuits in $Spec(N_1)$, LS_TI is scalable (if one keeps the granularity of $Spec(N_1)$ small.)

# 6. COMPLEXITY OF EXTERNAL EC OF CIRCUITS WITH COMMON SPECIFICATION

Let $N_1$ be a circuit with specification $Spec(N_1)=\{N_1^1,..,N_1^k\}$. Let $N_2$ be a circuit with specification $Spec(N_2)=\{N_2^1,..,N_2^k\}$. produced from $N_1$ by either by EC_TE or EC_TI. In this section, we discuss the complexity of external EC of $N_1$ and $N_2$.

## 6.1 EC of circuits produced by LS_TE

In [8], a top commercial tool was used for "external" EC of circuits with a common specification (Table 2 of [8]). These results showed that even for circuits with a common specification of small granularity, their EC was too hard for that tool (even with a 10 hour time limit). On the other hand, all examples were solved by EC_TE within 1-2 minutes.

One can always pick circuits $N_1$ and $N_2$ with a common specification that will "break" current EC algorithms. The reason is that an external checker inevitably makes implicit assumptions that can be easily broken. For example, algorithms based on computing cut-points make an assumption that $N_1$ and $N_2$ have functionally equivalent internal points. However, if $N_2$ is produced from $N_1$ by LS_TE, $N_1$ and $N_2$, in general, have no functionally equivalent points. Algorithms based on BDD computation make an implicit assumption that $N_1$ and $N_2$ have a small width while LS_TE can be used for optimization of circuits of arbitrary width. EC based on recursive learning [11] assumes that implications relating points of $N_1$ and $N_2$ can be obtained inductively by a computation of small "recursion depth". This assumption can be easily broken as well. The method of [12] also makes a breakable assumption that $N_1$ and $N_2$ do not have a large number of reconvergent fan-outs.

In terms of proof sizes (computed with respect to a concrete proof system like resolution), the problem with existing (and most likely any) external equivalence checkers is as follows. It may well be the case that any proofs of equivalence different from the ones generated by LC_TE are much "longer". On the other hand, to find a proof generated by LC_TE one needs to build partitions $Spec(N_1)$ and $Spec(N_2)$, which is very hard. The reason is that finding a pair of subcircuits $N_1^i$, $N_2^i$ that are toggle equivalent requires testing $\approx|N_1|^{p1}*|N_2|^{p2}$ pairs of subcircuits where $p_i$, $i$=1,2 is the granularity of $Spec(N_i)$.

## 6.2 EC of circuits produced by LS_TI

Although external verification of circuits built by LS_TE looks infeasible, verification of circuits produced by LS_TI is "even harder". The reason is as follows. Suppose that $N_2$ with specification $Spec(N_2)$

is produced from $N_1$ with specification $Spec(N_1)$ by LS_TE. Suppose an external equivalence checker somehow managed to find subcircuits $N_1^i$ and $N_2^i$ that are toggle equivalent. Then, it has to decide whether this toggle equivalence is "accidental" or $N_1^i$ and $N_2^i$ are subcircuits of $Spec(N_1)$ and $Spec(N_2)$. If $N_1^i$ and $N_2^i$ are toggle equivalent "accidentally", then one cannot use outputs of $N_1^i$ and $N_2^i$ as "cut-points" to find subcircuits that are toggle equivalent in terms of previous cut-points (because the wrong choice of cut-points leads to false negatives). However, it is conceivable that toggle equivalence of subcircuits of $N_1$ and $N_2$ is a "rare" occasion and so $N_1^i$ and $N_2^i$ are subcircuits of $Spec(N_1)$, $Spec(N_2)$ with some reasonable probability.

The situation with LS_TI is vastly different. If $N_2$ with specification $Spec(N_2)$ is obtained from $N_1$ with specification $Spec(N_1)$ by LS_TI, any method of finding $Spec(N_1)$ and $Spec(N_2)$ faces huge false negative problem. Indeed, if $N_1^i \leq N_2^i$ holds for some subcircuits of $N_1$ and $N_2$, then $N_1^i \leq N'^i_2$ also holds if $N'^i_2$ is a subcircuit of $N'^i_2$ such that the outputs of $N'^i_2$ form a cut of $N_2^i$. (A cut of $N_2^i$ cannot toggle "less" than the set of outputs of $N_2^i$.) Besides, $N_1^i \leq N''^i_2$ also holds if $N_2^i$ is a subcircuit of $N''^i_2$ and the set of outputs of $N''^i_2$ contains all the outputs of $N_2^i$ (adding more outputs to $N_2^i$ only increases toggling.) So, the number of pairs of subcircuits $N_1^i$ , $N_2^i$ for which $N_1^i \leq N_2^i$ holds is, in general, astronomical. So picking the "right" pair of subcircuits $N_1^i$, $N_2^i$ is extremely unlikely and hence finding $Spec(N_1)$ and $Spec(N_2)$ looks even "more impossible" than in the case of LS_TE.

## 7. CONCLUSION

In this paper, we discuss how "external" equivalence checkers can be affected by the appearance of new powerful logic synthesis procedures. Our results imply that the increasing power of synthesis procedures may make external equivalence checking problematic if not impossible.

## 8. REFERENCES


[1] C.L.Berman. *Circuit width, register allocation, and ordered binary decision diagrams.* IEEE Trans. on CAD. Vol 10:8, 1991, pp. 1059-1066.

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

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

[4] R.Bryant. *Graph-Based Algorithms for Boolean Function Manipulation*. IEEE Trans. on Computers, Vol. C - 35, No. 8, August, 1986, pp. 677 - 691.

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

[6] R. Drechsler and S. Horeth. Gatecomp: *Equivalence Checking of Digital Circuits in an Industrial Environment*, International Workshop on Boolean Problems, pp. 195-200, 2002.

[7] *Electronic Design Automation For Integrated Circuits Handbook*, by L.Lavagno, G.Martin, and L.Scheffer, Volume 2, Chapter 4, *Equivalence Checking*, by F. Somenzi and A. Kuehlmann.

[8] E.Goldberg. *On Equivalence Checking and Logic Synthesis of Circuits with a Common Specification.* Proceedings of GLSVLSI, Chicago, April 17-19, 2005, pp.102-107, (http://eigold.tripod.com/papers/glsvlsi-2005.pdf ).

[9] A. Gupta, P.Ashar. *Integrating a Boolean Satisfiability Checker and BDDs for Combinational Equivalence Checking*. VLSI Design 1998. pp. 222-225.

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

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

[12] H.Kwak, I.Moon, J.Kukula, T.Shiple. *Combinational equivalence checking through function transformation*, ICCAD-2002, pp. 526-533.

[13] I.-H.Moon, C. Pixley. *Non-miter-based Combinational Equivalence Checking by Comparing BDDs with Different Variable Orders.* FMCAD 2004, 144-158

[14] J. Moondanos, C.-J. H. Seger, Z.Hanna, D. Kaiss. *CLEVER: Divide and Conquer Combinational Logic Equivalence VERification with False Negative Elimination*. CAV-2001,pp. 131-143.

[15] E.M. Sentovich et. al. *SIS: A system for sequential circuit synthesis*. Technical report, University of California at Berkeley, 1992. Memorandum No. UCB/ERL M92/41.

[16] S.Sinha , S.Khatri, R.Brayton, A. Sangiovanni-Vincentelli. *Binary and multi-valued SPFD-based wire removal in PLA networks*, ICCD-2000, pp. 494-503.

[17] E.Goldberg,K.Gulati,S.Khatri *Toggle Equivalence Preserving (TEP) logic synthesis*. IWLS-2007, San Diego 2007 (http://eigold.tripod.com/papers/iwls-2007-tep.pdf)