Toggle Equivalence Preserving Logic Synthesis

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

CDNL-TR-2005-0912
September 2005

Eugene Goldberg (Cadence Berkeley Labs), egold@cadence.com
Kanupria Gulati (Texas A&M University), kanu.gulati@gmail.com

Abstract. Earlier, a theory was introduced that enabled a method of scalable logic synthesis. In this method, given a circuit N_1 and its partition into subcircuits N_1^i, i=1,..k, an optimized circuit N_2 functionally equivalent to N_1 is built by replacing each subcircuit N_1^i with a toggle equivalent counterpart N_2^i. To implement this method in practice, one needs a procedure that, given a multi-output subcircuit N_1^i, builds another multi-output subcircuit N_2^i that is toggle equivalent to N_1^i. In this report, we introduce such a procedure and test it on MCNC benchmarks. Experiments show great potential of the new method of logic synthesis.

1. Introduction

A generic problem of electronic CAD is to implement a Boolean function f as a combinational circuit that is optimized with respect to a cost function. The size of combinational blocks may grow very large in modern chips, so CAD needs scalable algorithms of logic optimization. Generally speaking, a scalable algorithm of an optimization problem has to satisfy the following two properties: a) it should have polynomial (in practice, close to linear) complexity, b) it should pick a solution out of a "large" subspace of the search space. The second condition is important because otherwise, one can use a "super efficient" algorithm outputting the initial instance as it is.

Today's logic optimization procedures do not satisfy the definition of scalability above. The problem is that their good performance is obtained by the "uncontrollable" reduction of the set of candidate circuits. For example, if a synthesis procedure uses SAT to check the validity of a logic transformation, a bound on the number of backtracks may be imposed. This results in an unpredictable depleting of the set of circuits to be considered by this procedure.

In [1] a new method of logic synthesis was introduced. Suppose N_1 is a single-output circuit to be optimized and a partitioning of N_1 into subcircuits N_1^i, i=1,..,k is specified (see Figure 1). The main idea of the method is to optimize circuit N_1 by replacing each subcircuit N_1^i with a *toggle equivalent* counterpart N_2^i,i=1,..,k. (We will refer to this method as TEPLS, which stands for Toggle Equivalence Preserving Logic Synthesis). Subcircuits N_2^i are connected exactly as N_1^i forming a circuit N_2 that is *functionally equivalent* to N_1.

The main appeal of TEPLS is that a) it gives an efficient procedure for finding optimized circuit N_2 and b) the set of candidate circuits is well-defined and huge. In other words, TEPLS is *scalable*. To realize how big the set of candidate circuits is, it suffices to say that the number of k-output functions toggle equivalent to a k-output function $\boldsymbol{f_i}$ (implemented by subcircuit N_1^i) is $\approx(2^k)!$. For k=1,2,3,4,5 this number equals 2, 24, $4*10^4$, $2.1*10^{13}$, and $2.6*10^{35}$ respectively. So TEPLS enjoys enormous flexibility even if N_1 is partitioned into very small subcircuits N_1^i.

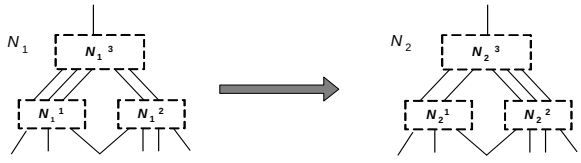

Figure 1. Optimization of circuit N_1 by TEPLS

On the other hand, the TEPLS procedure of [1] is linear in the number of subcircuits N_1^i and exponential in the size of N_1^i, N_2^i. So if the size of subcircuits N_1^i and N_2^i is

bounded by a constant (which still leaves a huge number of candidate circuits), TEPLS has *linear* complexity. This efficiency is due to the fact that when replacing a subcircuit N_1^i with a subcircuit N_2^i, their toggle equivalence is maintained only *locally* in terms of input variables of N_1^i and N_2^i, related by so-called correlation functions [1]. These correlation functions can be efficiently computed "inductively".

Unfortunately, [1] did not provide a specific procedure that, given a subcircuit N_1^i, would build a toggle equivalent subcircuit N_2^i. (We will refer to it as Toggle Equivalence Preserving (TEP) procedure). ***The main contribution*** of this report is the introduction of such a procedure. Since *single-output* Boolean functions f_1 and f_2 that are toggle equivalent are also *functionally equivalent* [1], our TEP-procedure can be also used for "regular" logic synthesis. In experiments we compared our TEP-procedure with SIS on MCNC benchmarks. Surprisingly, for many single-output circuits (no advantage of using toggle equivalence) our TEP-procedure found much better solutions than SIS. For multiple output circuits (where using toggle equivalence is advantageous) the TEP-procedure gave much more dramatic improvements over SIS. Finally, we used our TEP procedure in TEPLS. Experiments showed that TEPLS can achieve great gate reduction in comparison to SIS.

This report is structured as follows. In Sections 2,3,4,5 we essentially recall the TEPLS method of [1] and give a few new definitions (like implication of toggling). Section 6 describes our TEP procedure. In Section 7, we give experimental results. In Section 8 we briefly discuss the relation of TEPLS and SPFDs [3][4]. Section 9 describes how one can use TEPLS for peephole optimization. Finally, some conclusions are made in Section 10.

2. Toggle equivalence of Boolean functions

In this section, we recall the notion of toggle equivalence and its properties. All the propositions given in this report are either proven in [1] or can be easily derived from them.

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 if $f^* = f(\ldots, x=0,\ldots) \vee f(\ldots, x=1,\ldots)$.

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 $Sat(v(N)) = (\sim x_1 \vee \sim x_2 \vee y) \wedge (x_1 \vee \sim y) \wedge (x_2 \vee \sim y)$.

Proposition 3. Let N_1 and N_2 be toggle equivalent and A_1, A_2 be the sets of their output variables. Let function $K^*(A_1,A_2)$ be obtained from $Sat(v(N_1)) \wedge Sat(v(N_2))$ by existentially quantifying away the variables of $v(N_1) \cup v(N_2)$ except those of $A_1 \cup A_2$. The function $K^*(A_1,A_2)$ implicitly specifies the one-to-one mapping K between output vectors produced by N_1 and N_2.

3. Implication of toggling

In this section, we introduce the notion of implication of toggling and describe how toggle equivalence and implication of toggling can be tested.

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

Definition 6. A multi-output function $\boldsymbol{f_1(x_1,..,x_n)}$ **strictly implies toggling of** a multi-output function $\boldsymbol{f_2(x_1,..,x_n)}$ if $\boldsymbol{f_1}$ implies toggling of $\boldsymbol{f_2}$ and there is a pair of assignments $\boldsymbol{x'}$, $\boldsymbol{x''}$ to the variables of X such that $\boldsymbol{f_1(x') = f_1(x'')}$ while $\boldsymbol{f_2(x') \neq f_2(x'')}$.

Remark 1. We will **denote** by $\boldsymbol{f_1} \leq \boldsymbol{f_2}$ (respectively $\boldsymbol{f_1} < \boldsymbol{f_2}$) the fact that function $\boldsymbol{f_1}$ implies toggling of **(**respectively strictly implies toggling of) $\boldsymbol{f_2}$. We will **denote** by $N_1 \leq N_2$ (respectively $N_1 < N_2$) the fact that 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 f_1 and f_2 are toggle equivalent iff $f_1 \le f_2$ and $f_1 \le f_2$.

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 $A=\{a_1,\ldots,a_m\}$ and $B=\{b_1,..,b_k\}$ be the sets of output variables of N_1 and N_2 respectively. Then $N_1 \le N_2$ holds iff the function $S = H(v(N_1), v(N_2)) \wedge H(v(N^*_1), v(N^*_2)) \wedge Neq(A, A^*) \wedge Eq(B, B^*)$ 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 $A^*=\{a^*_1,\ldots,a^*_m\}$ and $B^*=\{b^*_1,..,b^*_k\}$ respectively. The function $H(v(N_1), v(N_2))$ is equal to $Sat(v(N_1)) \wedge Sat(v(N_2))$. The value of $Eq(\boldsymbol{b}, \boldsymbol{b^*})$ where $\boldsymbol{b}$ and $\boldsymbol{b^*}$ are assignments to B and B^* respectively is equal to 1 iff $\boldsymbol{b}=\boldsymbol{b^*}$. The function $Neq(A, A^*)$ is the negation of $Eq(A, A^*)$.

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

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

4. Correlation function

In this section, we use the notion of correlation function to extend definitions of toggle implication and toggle equivalence to the case when functions f_1 and f_2 have different sets of variables.

Definition 7. 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 $Q^X \subseteq \{0,1\}^{|X|}$ and $Q^Y \subseteq \{0,1\}^{|Y|}$ such that $Cf(X,Y)$ specifies a bijective mapping $M: Q^X \to Q^Y$. Namely $Cf(\boldsymbol{x}, \boldsymbol{y})$=1 iff $\boldsymbol{x} \in Q^X$ and $\boldsymbol{y} \in Q^Y$ and $\boldsymbol{y} = M(\boldsymbol{x})$.

Informally, $Cf(X,Y)$ is a correlation function if it specifies a bijective mapping between a subset Q^X of $\{0,1\}^{|X|}$ and a subset Q^Y of $\{0,1\}^{|Y|}$.

Let $f_1(X)$ and $f_2(Y)$ be two multi-output Boolean functions where $X=\{x_1,\ldots,x_k\}$ and $Y=\{y_1,\ldots,y_p\}$ are sets of their variables. (Note, that f_1 and f_2 may have different number of variables.). Let $Cf(X, Y)$ be a correlation function relating variables of f_1 and f_2. Then one can introduce notions of toggle equivalence and toggle implication for f_1 and f_2. The only difference from definitions and results listed in Sections 2 and 3 is that now one should consider only assignments that satisfy $Cf(X,Y)$.

For example, f_1 and f_2 are said to be toggle equivalent, if for any pair of pairs $(\boldsymbol{x}, \boldsymbol{y})$ and $(\boldsymbol{x'}, \boldsymbol{y'})$ of input vectors such that $Cf(\boldsymbol{x}, \boldsymbol{y})=Cf(\boldsymbol{x'}, \boldsymbol{y'})$=1, it is true that $f_1(\boldsymbol{x}) \ne f_1(\boldsymbol{x'}) \Leftrightarrow f_2(\boldsymbol{y}) \ne f_2(\boldsymbol{y'})$. The mapping between output vectors produced by toggle equivalent circuits N_1 and N_2 (implementing functions f_1 and f_2 respectively), can be obtained from $Sat(v(N_1)) \wedge Sat(v(N_2)) \wedge Cf(X,Y)$ by existentially quantifying away all the variables of $v(N_1) \cup v(N_2)$ except output variables of N_1 and N_2. The other results and definitions of Sections 2 and 3 can be modified in a similar manner.

5. Toggle equivalence preserving logic synthesis

In this section, we recall the procedure of toggle equivalence preserving logic synthesis (TEPLS) introduced in [1]. The pseudocode of the TEPLS procedure is shown in Figure 2.

```
1 Synthesize(N1, Partition(N1),cost_function) {
2   for (i=1; i <= k ; i++) {
3       Cf_inp(N1^i, N2^i)= correlation_func(N1,N2,i);
4       N2^i = synth_toggle_equivalent(N1^i, Cf_inp,cost_function)
5       Cf_out(N1^i, N2^i) = exist_quantify(N1^i,N2^i, Cf_inp);   }
6 return(N2,Partition(N2))}
```

Figure 2. Pseudocode of TEPLS procedure

Following [1] we also assume that $Partition(N_1) = \{N_1^1,..,N_1^k\}$ (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_2^j. $Partition(N_1)$ is called ***topological*** if G is acyclic.) Since $Partition(N_1)$ is topological, one can assign levels to subcircuits N_1^i. Similar to [1] we assume that if $i < j$ then $topological\_level(N_1^i) \le topologica\_level(N_1^j)$. In other words, subcircuits N_1^i, i=1,..,k are processed by the TEPLS procedure in topological order, from inputs to outputs.

Let us consider how the TEPLS procedure works by the example of Figure 1. The procedure starts with subcircuit N_1^1 and recover the correlation function $Cf_{inp}(N_1^i, N_2^i)$ relating inputs of N_1^1 and 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, the correlation function is just a conjunction of equality functions relating corresponding inputs of N_1^1 and N_2^1. (This correlation function just "identifies" 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 $Cf_{out}(N_1^1, N_2^1)$ 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 $Cf_{out}(N_1^1, N_2^1)$ is a correlation function.

Then, the TEPLS procedure processes subcircuit N_1^2 in the same manner, generating a toggle equivalent subcircuit N_2^2 and the correlation function $Cf_{out}(N_1^2, N_2^2)$. 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 correlation function $Cf_{inp}(N_1^3, N_2^3)$ relating inputs of N_1^3 and subcircuit N_2^3

(synthesized in line 4) equals $Cf_{out}(N_1^1, N_2^1) \wedge Cf_{out}(N_1^2, N_2^2)$. (We take the conjunction of $Cf_{out}(N_1^1, N_2^1)$ and $Cf_{out}(N_1^2, N_2^2)$ because outputs of N_1^1 and N_1^2 feed inputs of N_1^3). It is not hard to show that a conjunction of correlation functions is a correlation function too. (So $Cf_{inp}(N_1^3, N_2^3)$ is a correlation function indeed.)

Since N_1^3 has only one output, subcircuits N_1^3 and N_2^3 (under constraints on input assignments specified by $Cf_{inp}(N_1^3, N_2^3)$) are functionally equivalent modulo complement. So the circuit N_2 consisting of subcircuits N_2^1, N_2^2, N_2^3 is functionally equivalent to N_1 (modulo complement).

6. A procedure for building a toggle equivalent circuit

In this section, we introduce a toggle equivalence preserving (TEP) procedure. This procedure (used in line 4 of Figure 2) is a cornerstone of TEPLS. Given a subcircuit N_1^i and correlation function Cf_{inp} relating inputs of N_1^i and a future subcircuit N_2^i, the TEP procedure actually builds N_2^i that is toggle equivalent to N_1^i. For the sake of simplicity, in this chapter, we drop the superscript *i* from N_1^i, N_2^i. Besides, we will assume that N_1^i and N_2^i have identical sets of input variables. (The extension to the case of different input variables related by a correlation function can be easily done as described in Section 4.)

The TEP procedure is based on the following observation. Suppose that N is a Boolean circuit and C' and C'' are two "topologically ordered" cuts (see Figure 3). Namely, no path P from an input to an output of N can "cross" C'' before C' (but C' and C'' may have common points). Let N' and N'' be the subcircuits of N consisting of the gates located between the primary inputs and the cut C' or C'' respectively. Then $N'' \leq N'$. In other words, if for a pair of input vectors a point of C'' toggles, there has to be at least one point of C' that toggles too.

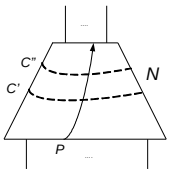

Figure 3. Two cuts

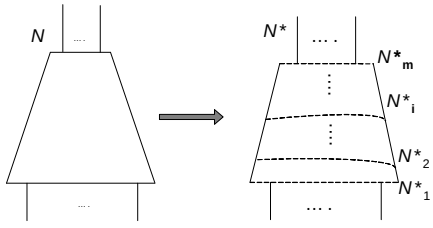

Figure 4. Sequence of circuits built by TEP procedure

Let N be a Boolean circuit. To build a circuit N^* that is toggle equivalent to N, the TEP procedure constructs a sequence of circuits $N^*_1,.., N^*_m$ (see Figure 4.) N^*_1 is an "empty" circuit consisting only of inputs of N while N^*_m is the final circuit N^* that is toggle equivalent to N. Circuits are built in such a way that for each N^*_i, i=2,..,m the following invariant holds: $N \leq N^*_i < N^*_{i-1}$. Informally speaking, every circuit N^*_i toggles at least as much as N and *strictly* less than all the circuits N^*_j, $j < i$. That $N \leq N^*_1$, trivially follows from the fact that the set of inputs of N forms a cut of N. Since every next circuit N^*_i "looses" a toggle in comparison to N^*_{i-1}, the sequence $N^*_1,.., N^*_m$ eventually converges to a circuit N^* toggle equivalent to N.

```
    generate_toggle_equivalent_circuit(N) {
1   if (constant(N)) return('constant');
2   N^curr = ∅;
3   while (true) {
4     iff (N^curr ≤ N )    return(N^curr);
5     N^curr = discard_toggles(N^curr, N)
6     N^curr = remove_redundant_outputs(N^curr); }
```

Figure 5. Pseudocode of TEP procedure

The pseudocode of the TEP procedure is shown in Figure 5. The sequence of circuits mentioned above is built in a loop (lines 3-6). This sequence starts with an "empty" circuit N^{curr} (line 2). In the loop, first, it is checked if $N^{curr} \leq N$ holds. If so, then N^{curr} is toggle equivalent to N (because $N \leq N^{curr}$ by construction). Otherwise, a new circuit N^{curr} is generated by the function *discard_toggles* (line 5) such that $N \leq N^{curr(new)} < N^{curr(old)}$. Finally, redundant outputs of N^{curr} are removed (line 6). (An output of N^{curr} is redundant if after its removal from N^{curr}, the inequality $N \leq N^{curr}$ still holds.)

```
discard_toggles(N^curr, N) {
1  R* = toggle_difference (N^curr, N ); /*  N^curr ≤ N  ?  */
2  (N', R) = remove_toggles(R*, N^curr);
3  D = toggle_difference(N, N');  /*  N  ≤ N' ? */
4  N'' = add_toggles(R, D, N', N^curr);
5  return(N'');}
```

Figure 6. Pseudocode of *discard_toggles*

The pseudocode of the function *discard_toggles* is shown in Figure 6. This procedure returns circuit N'' such that $N \leq N'' < N^{curr}$. The circuit N'' is built in two steps. First, a circuit N' such that $N' < N^{curr}$ is obtained from N^{curr} (lines 1-2). At this point, the inequality $N \leq N'$ may not hold. Second, the circuit N' is transformed into the final circuit N'' (lines 3-4).

In line 1, the set R^* of "redundant" toggles of N^{curr} is computed. As we mentioned in Section 3, checking if $N^{curr} \leq N$, reduces to testing the satisfiability of the function S defined there. To find the redundant toggles of N^{curr} one needs to enumerate all the assignments satisfying S. (However if the set R^* is too large one can use a manageable subset of R^*.) The function *remove_toggles* produces circuit N' by adding an AND gate to N^{curr}. It also computes a subset R of R^* consisting of the toggles that are *actually* removed after adding the AND gate above. The idea of toggle removal is illustrated in Figure 7 a).

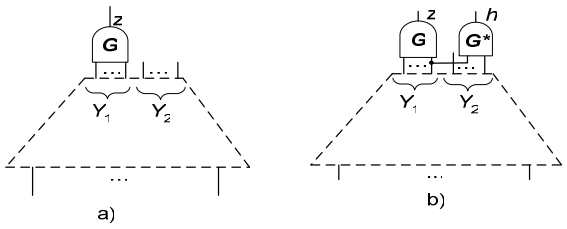

Figure 7. a) Toggle removal; b) Toggle "addition"

Let Y be the set of output variables of N^{curr}. Let $t_1=(\boldsymbol{y}, \boldsymbol{y}')$ be a toggle of N^{curr} from R^*. Let Y_1 (respectively Y_2) be the subset of Y corresponding to the components of $\boldsymbol{y}$ and $\boldsymbol{y}'$ that are different (respectively identical). That is Y_1 (Y_2) correspond to the toggling (non-toggling) outputs of N^{curr}. Then one can always add an AND gate G (see Figure 7 a) whose inputs are fed by the outputs Y_1 of N^{curr} such that $G(\boldsymbol{y})=G(\boldsymbol{y}')$. Then the circuit N' (obtained from N^{curr} after adding G) with the set of output variables $\{z\} \cup Y_2$ (here z specifies the output of G) does not toggle for any pair of input vectors producing the toggle $t_1=(\boldsymbol{y}, \boldsymbol{y}')$ in N^{curr}.

Unfortunately, adding gate G may remove some "useful" toggles. (A toggle of N' is useful if its removal breaks inequality $N \le N'$.) So, one needs to reintroduce them by adding gates to the circuit N' and transforming it into a circuit N''. The set D of toggles to be reintroduced is computed by the same function *toggle_difference* (line 3). However, in contrast to R^*, the set D has to be computed exactly (to guarantee that $N \le N''$.) Let toggle $t_2 = (\boldsymbol{h},\boldsymbol{h}')$ be a useful toggle of N^{curr} removed from N'. The set of outputs of N^{curr} toggling in t_2 is a subset of the set Y_1. (If an output of Y_2 toggled in t_2, then any pair of input vectors producing toggle t_2 in N^{curr} would make N' toggle too.) It can be shown , that one can always reintroduce a toggle t_1 of the set D by adding an AND gate (see Figure 7 b), *without reintroducing* a toggle t_1 of R. In other words, the *discard_toggles* procedure *always* returns a circuit N'' (that becomes new N^{curr}) such that $N \le N'' < N^{curr}$.

Procedures *remove_toggles* and *add_toggles* are heuristic. *remove_toggles* picks a gate that removes a maximal set R of toggles from R^*. *add_toggles* picks an AND gate so as to minimize the number of reintroduced toggles of R while maximizing the number of reintroduced toggles of D. *add_toggles* keeps adding gates (using the same cost function) until all toggles of D are reintroduced.

7. Experimental Results

In this section, we compare our TEP procedure and TEPLS based on this TEP procedure with SIS [2]. In all experiments, we compared the results of optimization by SIS (first running script.rugged and then technology decomposition to obtain a circuit of two-input AND gates) and by TEP or TEPLS. After running TEP or TEPLS (that generate a circuit of AND gates) we also used technology decomposition of SIS to make sure that all AND gates of the resulting circuit had only two inputs. The results are given in Tables 1,2,3 which are just samples of the vast experimental data we got.

In Table 1 results of running TEP and SIS on single-output subcircuits obtained from some MCNC benchmarks are shown. The objective of using single-output benchmarks was to show that our TEP procedure can be used in "regular" logic synthesis.

Circuit (output)	#in-puts	SIS #gates	TEP #gates
5xp1 (9)	7	11	**6**
alu2(0)	8	102	**98**
apex3(44)	8	21	**14**
b12(1)	7	14	**8**
example2(24)	7	11	**6**
i7(0)	5	12	**4**
misex1(5)	6	22	**18**
ttt2(1)	8	36	**26**
sqrt8ml(2)	8	29	**17**
5xp1(0)	7	**19**	23
ex5p(43)	8	**43**	54
f51m(2)	7	**30**	32

Table 1. Results of TEP procedure on single-output circuits

Each circuit of Table 1 is the subcircuit feeding an output of a benchmark. The first column of Table 1 shows the name of the benchmark and the output used. The second column gives the number of input variables. Third and fourth columns of Table 1 give the size of resulting circuits (in the number of gates) obtained by SIS and the TEP procedure. As one can see, in many cases, the TEP procedure managed to obtain much smaller circuits than SIS.

Table 2 contains results of SIS and the TEP procedure on some multi-output MCNC circuits. The third column contains the number of outputs in circuits used. The fourth column contains numbers of outputs in toggle equivalent circuits generated by the TEP procedure. Fifth and sixth columns contain the size of circuits (in the number of gates) obtained by SIS and the TEP procedure. These results show that in some cases TEP can dramatically reduce circuit size. For some circuits (5xp1, f51m) TEP removed all gates. This means that these circuits produce different outputs for different input vectors. So an "empty" circuit just copying each input to its output is toggle equivalent to the original circuit.

Of course, replacing a circuit N_1 with a toggle equivalent counterpart N_2 may make the surrounding logic more complex. So even if the circuit N_2 is much smaller than N_1, the overall gain (if any) may be much more moderate. However, Table 3 gives a proof that employing the flexibility of toggle equivalence is very beneficial.

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**
cm138a	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

Table 2. Results of TEP procedure on multi-output circuits

Level 1	Le-vel 2	SIS #gates	TEPLS #gates
squar5	C_1	18	**16**
	C_2	28	**14**
	C_3	37	**19**
rd84	M_1	104	**49**
	M_2	123	**51**
	M_3	138	**49**

Table 3. Results of TEPLS

It contains the results of using TEPLS to optimize 6 circuits. Each of these 6 circuits consisted of an MCNC benchmark (square5 or rd84) whose outputs fed a single-output circuit (denoted in Table 3 as C_i,i=1,2,3 or M_i=1,2,3.) Hence, each of the 6 circuits had only one output.

We optimized these 6 circuits using SIS and the TEPLS algorithm (shown in Figure 2) that employed our TEP procedure. Since the 6 circuits of Table 3 had only one output, each circuit produced by TEPLS was functionally equivalent to the original one modulo negation. TEPLS was given the "natural" partition of the initial circuit N_1 into subcircuits N_1^1, N_1^2 where N_1^1 was a benchmark circuit and N_1^2 was C_i or M_i. The results of optimization are given in columns 3 and 4. They clearly show huge potential of TEPLS. Even for very small circuits of Table 3, TEPLS got a dramatic improvement over SIS. This improvement was achieved by replacing first level circuit N_1^1 with a much smaller toggle equivalent subcircuit N_2^1. TEPLS managed to "preserve" this gain when generating subcircuit N_2^2 that was toggle (and functionally) equivalent to the second level subcircuit N_1^2.

8. TEPLS and SPFDs

In this section, we briefly discuss the relation of TEPLS and a method of computing SPFDs from [3][4]. In that method, a relation (called Set of Pairs of Functions to be Distinguished or SPFD for short) is computed for a circuit N_1 in the backward movement, i.e. from outputs to inputs. Then, in the forward movement, gates (or subcircuits) satisfying these relations are built, which results in obtaining an optimized circuit N_2. The main disadvantage of the method for computing SPFDs from [3][4] is that it is *unscalable*. (Note that here we are talking not about SPFDs as a notion, but about a particular method of their computation. SPFDs as a notion is just a way to represent informational content of a circuit node.). This unscalability is due to the fact that when propagating SPFDs from outputs to inputs one may have to compute the relations between points located arbitrarily far from each other in the circuit N_1. So the logic involved in this computation may comprise an arbitrarily large part of N_1. One more flaw of the method for computing SPFDs from [3][4] is that it essentially follows *the gate level structure* of circuit N_1 to be optimized (while TEPLS preserves only the topology of N_1 in terms of subcircuits).

What SPFDs and TEPLS have in common is that they both use the "optimization as communication" paradigm. By replacing subcircuits N_1^i of N_1 with their toggle equivalent counterparts N_2^i, TEPLS essentially "redistributes" complexity between subcircuits. (In other words, subcircuits "talk" to each other through particular choices of encodings.) A similar paradigm is used in the method of SPFDs at the "gate level".

9. TEPLS and peephole optimization

The TEPLS procedure described in Section 5 requires the circuit N_1 to be optimized to be partitioned into subcircuits. In this section, we give an example of how TEPLS can be used even if partition of N_1 into subcircuits is not specified (see Figure 8). Suppose that N' is a two-output subcircuit of N_1. (This means that every path from an input to an output of N_1 that "crosses" N' goes through one of these two outputs.) One can use our TEP procedure to replace N' with a toggle equivalent two-output subcircuit N''. Let h_1 and h_2 be the only gates connected to outputs of N'. To guarantee that the new circuit N_2 is functionally equivalent to N_1 one can always replace gates h_1 and h_2 with subcircuits H_1 and H_2 such that $y' = y''$ and $z' = z''$. Note that since N'' "reencodes" output assignments of N', H_1 and H_2 need both outputs of N''.

The idea of such peephole optimization is as follows. If the number of outputs in N' is small and N'' is much smaller than N', then the size of N_2 should be noticeably smaller than that of N_1. (Because the size of subcircuits H_i fed by the outputs of N'' cannot be too large due the small number of ouputs in N'').

10. Conclusions

We introduced a procedure that optimizes a circuit by generating a toggle equivalent one. This procedure is a key element of TEPLS, a scalable method of logic synthesis.

Experiments show that the great flexibility of toggle equivalence indeed allows one to find very good solutions.

References

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

[2] 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.

[3] S.Sinha, R.K.Brayton. *Implementation and use of SPFDs in optimizing Boolean networks*. ICCAD-1998, pp. 103-110.

[4] S.Yamashita,H.Sawada,A.Nagoya. *A new method to express functional permissibiities for LUT based FPGAs and its applications.* ICCAD-1996, pp.254-261.

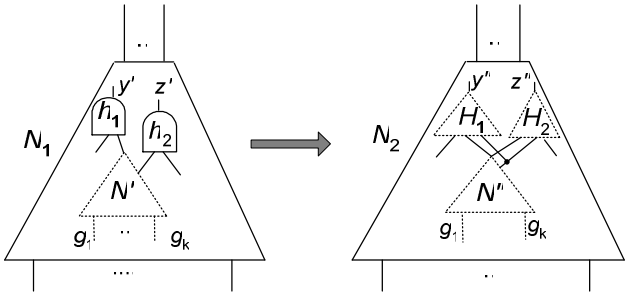

Figure 8. TEPLS used for peephole optimization