Logic synthesis preserving high-level specification

E.Goldberg (Cadence Berkeley Labs, USA),

Abstract. *In this paper we develop a method of logic synthesis that preserves high-level structure of the circuit to be synthesized. This method is based on the fact that two combinational circuits implementing the same "high-level" specification can be efficiently checked for equivalence. Hence, logic transformations preserving a predefined specification can be made efficiently. We introduce the notion of toggle equivalence of Boolean functions and show that toggle equivalence can be used for making gate level transformations that preserve a predefined specification. We describe a practical procedure for checking toggle equivalence of two Boolean circuits and give experimental data about its performance.*

1. Introduction

In a typical design flow, by the time a circuit is passed to a logic synthesis procedure, the "high-level" structure of this circuit is lost. Suppose, for example, that a combinational circuit is initially described as a network of multi-valued blocks. After encoding all the multi-valued variables, this network is replaced with a Boolean circuit that is optimized using a set of local transformations that ignore the original high-level structure of the circuit.

The flaw of the approach above is that in the case of a poor choice of encodings for multi-valued variables, a synthesis procedure using local transformations will not be able to "correct" these encodings. On the other hand, finding good encodings at the level of multi-valued blocks is hard and so the probability of generating bad encodings is high. A possible solution to the problem is to perform logic transformations that re-encode multi-valued variables "implicitly" at the gate level. We will call such transformations ***High-Level structure aware Logic Synthesis (HLLS)***). This is because re-encoding of multi-valued variables implicitly, essentially means synthesizing a circuit that is a different implementation of the same "specification" as the original circuit. An HLLS procedure can be used as an extra optimization step taken before using logic synthesis based on local transformations.

In this paper we introduce a method of HLLS for combinational circuits. To design an HLLS procedure one has to solve two problems. The first problem is to verify the correctness of logic transformations. A "regular" logic synthesis procedure makes local transformations so the equivalence checking of the original and optimized circuits is usually not an issue. On the other hand, an HLLS procedure performs "non-local" synthesis transformations, so verification of the correctness of such transformations may pose a problem. This problem was addressed in [3], [4] where it was shown that if two Boolean circuits have a common specification (CS), their equivalence checking is "easy". Informally, circuits N_1 and N_2 have a CS if they can be considered as two different implementations of a circuit S of multi-valued gates further referred to as blocks. (S is called a CS of N_1 and N_2). In [3][4] it was proven that given a CS S of circuits N_1 and N_2, there is an equivalence checking procedure whose complexity is linear in the number of blocks of S and exponential in the granularity of S (the "size" of the largest block of S). An example of circuits N_1,N_2 having a common specification of three blocks is shown in Fig. 1. The specification itself is shown on the left. Here $N_1(G_k)$ and $N_2(G_k)$ are subcircuits of N_1 and N_2 respectively implementing the same block of specification.

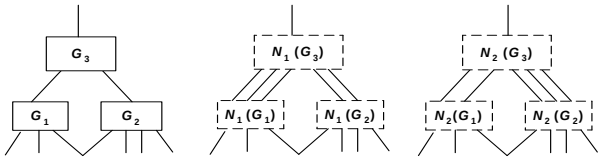

Figure 1 Circuits N_1 and N_2 with a common specification of three blocks

The second problem one has to solve in HLLS, is to find a way to preserve a predefined specification by making transformations at the gate level. Solving this problem is the focus of this paper. We introduce the notion of toggle equivalence of multi-output Boolean functions that is a generalization of regular functional equivalence. We show that circuits N_1 and N_2 have a CS if they can be partitioned into toggle equivalent subcircuits (see Section 4.). This result suggests a simple way to perform HLLS. Suppose that a specification S of circuit N_1 to be optimized is specified by partitioning the latter into subcircuits. Then, if we replace subcircuits of N_1 with "better" subcircuits that are toggle equivalent to replaced ones, we produce a circuit N_2 that implements the same specification S as N_1. To be viable, HLLS needs an efficient algorithm for checking toggle equivalence. We describe such an algorithm and demonstrate its efficiency on benchmark circuits .

The paper is structured as follows. In Section 2 we formally define the notion of a common specification. In

Section 3 we reformulate the algorithm of equivalence checking from [3] removing some redundancy. Section 4 introduces the notion of toggle equivalence of Boolean functions. The relation between the notions of toggle equivalence and common specification is shown in Section 5. In Section 6 we describe a method of HLLS. The relation of HLLS to other synthesis procedures is discussed in Section 7. The description of a procedure for checking toggle equivalence and its performance on benchmark circuits are given in Section 8. Finally, we draw some conclusions in Section 9.

2. Definition of common specification

In this section, we formally define the notion of a common specification of Boolean circuits. Let S be a single output combinational circuit of multi-valued blocks specified by a directed acyclic graph H. The sources and the sink of H correspond to primary inputs and the output of S. Each non-source node of H corresponds to a multi-valued block computing a multi-valued function of multi-valued arguments. Each node of n of H is associated with a ***multi-valued variable*** A. If n is a source of H , then the corresponding variable specifies values taken by the corresponding primary input of S. If n is a non-source node of S then the corresponding variable describes the values taken by the output of the block specified by n. If n is a source (respectively the sink), then the corresponding variable is called a ***primary input variable*** **(**respectively ***primary output variable)***. We will use the notation $C=G(A_1,A_2,\ldots, A_k)$ to indicate that a) the output of a block G is associated with a variable C; b) the function computed by the block G is $G(A_1,A_2,\ldots, A_k)$; c) only k nodes of H are connected to the node n in H and outputs of these nodes are associated with variables $A_1,A_2,\ldots, A_k$.

Denote by $D(A)$ the ***domain*** of the variable A associated with a node of H. The value of $|D(A)|$ is called the ***multiplicity*** of A. If the multiplicity of every variable A of S is equal to 2 then S is a ***Boolean circuit***.

Now we introduce the notion of a specification of a single output Boolean circuit N. Informally, a multi-valued circuit S is a specification of N if N can be obtained from S by picking proper encodings of internal variables of S. In the following exposition any multi-valued network is called a specification.

Definition 1. Let $D(A)=\{a_1,\ldots,a_t\}$ be the domain of a variable A of S. Denote by $q(A)$ a Boolean encoding of the values of $D(A)$ which is a mapping $q:D(A)\rightarrow\{0,1\}^m$ such that $a_i \neq a_j \Rightarrow q(a_i) \neq q(a_j)$. The value of $q(a_i)$, $a_i \in D(A)$ is called the ***code*** of a_i. Denote by $length(q(A))$ the number of bits in $q(a_i)$ i.e. the value of m. Denote by $v(A)$ the set of m ***coding Boolean variables***.

In this paper, we make the assumptions below about specifications and implementations

Assumption 1. A specification contains only one output. All primary input variables and the primary output variable of a specification are Boolean.

Assumption 2. Every gate of a Boolean circuit (implementation) has two inputs and one output. Every multi-valued block of a specification has only one output but the number of inputs of a block is not fixed.

Assumption 3. If A_i and A_j are two different variables of a specification , then $v(A_i) \cap v(A_j) = \varnothing$.

Definition 2. Let $C=G(A_1,A_2,..,A_n)$ be a block of specification S. Let $q(A_1),..,q(A_n),q(C)$ be encodings of variables $A_1,A_2,..,A_n$ and C respectively. A Boolean circuit N is said to ***implement the block G*** if N implements the completely specified Boolean function $q(C)=G(q(A_1),..,q(A_n))$ whose truth table is obtained from that of G by replacing values of $A_1..,A_n$,C with their codes.

Definition 3. Let S be a multi-valued circuit. A single output Boolean circuit N is said to ***implement the specification S***, if N can be built from S by the following two rules.

1) Each block G of S is replaced with its implementation (denote it by $N(G)$).

2) Let the output of block G_1 (specified by variable C) be connected to an input of block G_2 (specified by the same variable C) in S. Then the outputs of the circuit $N(G_1)$ are properly connected to inputs of $N(G_2)$. Namely, if a primary output of $N(G_1)$ connected to an input of $N(G_2)$ these input and output are specified by the same coding variable of $v(C)$

Remark 1. It is important to emphasize that the fact that a circuit N has a specification S does not necessary mean that N *is produced* from S by encoding it multi-valued variables. It just means that N *can* be produced from S.

Remark 2. Let N be an implementation of a specification S. Let p be the largest number of gates used in an implementation of a multi-valued block of S in N. We will say that S is a specification of ***granularity*** p for N.

Definition 4. Let N_1, N_2 be two functionally equivalent single output Boolean circuits. Let N_1, N_2 implement a specification S. Then S is called ***a common specification*** (CS) of N_1 and N_2.

Definition 5. Let S be a CS of N_1,N_2. Let p_1 (respectively p_2) be the granularity of S with respect to N_1 (respectively N_2). Then we will say that S is a CS of N_1,N_2 of ***granularity*** $p = max(p_1,p_2)$.

3. Equivalence checking with a known common specification

In this section, we recall the equivalence checking algorithm of [3] and give a slightly modified version of it.

Let N_1 and N_2 be Boolean circuits with a CS S. Let G_k be a block of S. We will denote by $N_1(G_k)$ and $N_2(G_k)$ the implementations of the block G_k in N_1 and N_2 respectively.

Definition 6. Let S be a CS of N_1 and N_2. The ***topological level*** of a block G_k in a specification S is the length of the longest path from a primary input of S to G_k. (The length of a path is measured in the number of blocks on it. The topological level of a primary input is assumed to be 0.) Denote by ***level(G_k)*** the topological level of G_k in S. Denote by $level(N_i(G_k))$ the topological level of implementation of block G_k in N_i, i=1,2 that is assumed to be equal to $level(G_k)$.

Definition 7. Let $N_1(G_k)$ and $N_2(G_k)$ be implementations of a multi-valued block G_k whose output is associated with variable C. Let $q_1(C)$ and $q_2(C)$ be encodings of the variable C used in implementations $N_1(G_k)$ and $N_2(G_k)$. Function $Cf(v_1(C),v_2(C))$ is called a ***correlation function*** of encodings $q_1(C),q_2(C)$ if

a) $Cf(z_1, z_2)$=1 for any assignment z_1 to $v_1(C)$ and z_2 to $v_2(C)$ such that $z_1=q_1(c)$ and $z_2=q_2(c)$ where $c \in D(C)$.

b) Otherwise $Cf(z_1, z_2)$=0.

Definition 8. Let N be a Boolean circuit. Denote by $v(N)$ be the set of Boolean variables associated with the output of gates of N . Denote by $Sat(v(N))$ 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 9. Let f be a Boolean function. We will say that function f^* is obtained from f by existentially quantifying away variable x if $f^* = f(\ldots,x=0,\ldots) \vee f(\ldots,x=1,\ldots)$.

In [3] it was shown that if N_1 and N_2 have a CS S, one can check them for equivalence in the time linear in the number of blocks of S and exponential in the granularity of S. The essence of that algorithm is to compute so-called filtering and correlation functions in topological order. Here we give a modified version of this algorithm. The modification is that we discard computation of filtering functions from the algorithm.

Here is an informal proof that computation of filtering functions is not necessary. Let C be the variable associated with the output of a block G_k of S. From definitions of filtering (denoted by Ff) and correlation functions given in [3] it follows that

$$Ff(v_1(C)) \wedge Ff(v_2(C)) \wedge Cf(v_1(C),v_2(C))= Cf(v_1(C),v_2(C)).$$

So filtering functions can be dropped from the proof of Proposition 7 of [3] used to formulate the main result i.e. Proposition 8. (The use of filtering functions makes sense though, if one relaxes the definition of a correlation function. However, in this paper we stick to the definition of a correlation function given in [3].)

In the modified algorithm, only correlation functions are computed in topological order of N_1 and N_2. The algorithm starts with block implementations $N_1(G_k),N_2(G_k)$ of level 1 then process implementations of level 2 and so on. Let $level(N_1(G_k))$=level($N_2(G_k)$)=1 (i.e. inputs of $N_1(G_k)$ and $N_2(G_k)$ are primary inputs of N_1 and N_2.) Let C be the variable associated with the output of G_k and $Cf(v_1(C),v_2(C))$ be the correlation function relating encodings $q_1(C)$ and $q_2(C)$. This function is obtained from the function $Sat(v(N_1(G_k)) \wedge Sat(v(N_2(G_k))$ by existentially quantifying away all the variables except the variables associated with the outputs of $N_1(G_k)$ and $N_2(G_k)$.

Suppose $level(N_1(G_k))=level(N_2(G_k))=k$ and the correlation functions have been computed for the implementations of levels less than k. Let the output of G_k be associated with variable C_k. Let the inputs of G_k be connected to the outputs of blocks $G_{k1},\ldots,G_{km}$ associated with variables $C_{k1},..,C_{km}$ respectively. Then the correlation function $Cf(v_1(C_k),v_2(C_k))$ is obtained from the function $Cf(v_1(C_{k1}),v_2(C_{k1})) \wedge .. \wedge Cf(v_1(C_{km}),v_2(C_{km})) \wedge Sat(v(N_1(G_k)) \wedge Sat(v(N_2(G_k)))$ by existentially quantifying away all the variables except ones associated with the outputs of $N_1(G_k)$ and $N_2(G_k)$. Eventually the correlation function $Cf(f_1,f_2)$ is computed where f_1 and f_2 are Boolean variables specifying the outputs of N_1 and N_2. If $Cf(f_1,f_2) = (f_1 \vee \sim f_2) \wedge (\sim f_1 \vee f_2)$ (which is the equivalence function), then circuits N_1 and N_2 are functionally equivalent.

Definition 10. Given two functionally equivalent Boolean circuits N_1, N_2, S is called the ***finest common specification*** if it has the smallest granularity p among all the CSs of N_1 and N_2.

The complexity of the described algorithm is exponential in the granularity of S (i.e. in the size of the maximal subcircuit $N_i(G_k)$). So to evaluate the complexity of equivalence checking of N_1 and N_2 correctly, one needs to know the finest CS of N_1 and N_2 or a CS whose granularity is close to that of the finest one.

4. Toggle equivalence of Boolean functions

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

Definition 11. 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***.

Remark 3. 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 12. Let $\boldsymbol{f}$ be a multi-output Boolean function of n arguments. Denote by $Part(\boldsymbol{f})$ the partition of the set $\{0,1\}^n$ into blocks $B_1,\ldots,B_k$ such that $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}')$ if $\boldsymbol{x}, \boldsymbol{x}'$ are in the same block and $f(\boldsymbol{x}) \neq \boldsymbol{f}(\boldsymbol{x}')$ if $\boldsymbol{x}, \boldsymbol{x}'$ are in different blocks.

Proposition 1. Let $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ be toggle equivalent. Then $Part(\boldsymbol{f_1})=Part(\boldsymbol{f_2})$ i.e. for each block B_i of $Part(\boldsymbol{f_1})$ there is a block B'_j of $Part(f_2)$ such that $B_i=B'_j$ and vice versa.

Proof. Assume the contrary i.e. $Part(\boldsymbol{f_1}) \neq Part(\boldsymbol{f_2})$. Then there is a block B_i of $Part(\boldsymbol{f_1})$ such that no block B'_j of $Part(\boldsymbol{f_2})$ is equal to B_i. Then only the two cases below are possible.

a) B_i contains at least two input vectors. Then there is a pair of vectors $\boldsymbol{x},\boldsymbol{x}'$ of block B_i such that they are in different blocks of $Part(\boldsymbol{f_2})$. This means that $f_1(\boldsymbol{x})=\boldsymbol{f_1}(\boldsymbol{x}')$ while $\boldsymbol{f_2}(\boldsymbol{x}) \neq \boldsymbol{f_2}(\boldsymbol{x}')$ i.e $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are not toggle equivalent. So we have a contradiction.

b) B_i contains only one input vector x. Let B'_j of $Part(\boldsymbol{f_2})$ contain vector x. Block B'_j also contains at least one more input vector because otherwise $B_i=B'_j$. So the block B'_j contains two input vectors that are in different blocks of $part(\boldsymbol{f_1})$. Hence $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are not toggle equivalent and we again have a contradiction □

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.

Proof. From Proposition 1 it follows that $Part(\boldsymbol{f_1})=Part(\boldsymbol{f_2})$. Since $\boldsymbol{f_1}$, $\boldsymbol{f_2}$ are Boolean functions, $Part(\boldsymbol{f_1})$ and $Part(\boldsymbol{f_2})$ contain two blocks each. So the only two alternatives are $\boldsymbol{f_1}=\boldsymbol{f_2}$ or $\boldsymbol{f_1}=\sim\boldsymbol{f_2}$.

Proposition 3. Let $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ be two multi-output Boolean functions of n Boolean variables such that $Part(\boldsymbol{f_1})$=Part$(\boldsymbol{f_2})$. Then $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are toggle equivalent.

Proof can be performed by reasoning in the same way as in the proof of Proposition 1.

Proposition 4. Let F be a multi-valued function of n Boolean variables. Let C be the multi-valued variable specifying the output of F. Let $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ be Boolean functions obtained from F by using encodings q_1 and q_2 (of possibly different length) for the values of C. Then $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are toggle equivalent.

Proof. According to Definition 1 different values of C are assigned different codes. So $Part(\boldsymbol{f_1})=Part(\boldsymbol{f_2})=Part(F)$ where $Part(F)=\{B_1,\ldots,B_k\}$ and B_i consists of all the input vectors for which F takes the same value of C. From Proposition 3 it follows that $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are toggle equivalent.

Proposition 5. 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 $Part(\boldsymbol{f_1}),Part(\boldsymbol{f_2})$ contain k blocks. 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 $\boldsymbol{x}$ is in the m-th block of $Part(\boldsymbol{f_1})$.

So far we have considered toggle equivalence of functions with identical sets of arguments. Below, the notion of toggle equivalence is extended to the case of Boolean circuits with different sets of arguments that are related by encoding functions.

Definition 13. Let $X=\{x_1,\ldots,x_n\}$ and $Y=\{y_1,\ldots,y_k\}$ be two disjoint sets of Boolean variables. Denote by $Enc(X,Y)$ a Boolean function satisfying the following two conditions

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 $Enc(\boldsymbol{x}, \boldsymbol{y})=Enc(\boldsymbol{x}', \boldsymbol{y})=1$.

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

The function Enc is called an ***encoding function***.

Remark 4. An encoding function can be viewed as specifying two different encodings of the same multi-valued variable. Indeed, let $V=\{\boldsymbol{x_1},\ldots,\boldsymbol{x_k}\}$ be the set of all assignments to variables of X such that $Enc(\boldsymbol{x_i}, \boldsymbol{y})=1$ for some $\boldsymbol{y}$. Let $W=\{\boldsymbol{y}_1,..,\boldsymbol{y}_m\}$ be the set of all assignments to variables of Y such that $Enc(\boldsymbol{x}, \boldsymbol{y}_i) = 1$ for some $\boldsymbol{x}$. It is not hard to see that from Definition 13, it follows that $|W|=|V|$ and there exists a "natural" bijective mapping between X and Y that relates a vector $\boldsymbol{x_i}$ of V and the vector $\boldsymbol{y_j}$ of W for which $Enc(\boldsymbol{x}_i, \boldsymbol{y}_j)=1$. Vectors $\boldsymbol{x_i}$ and $\boldsymbol{y_j}$ can be considered as codes of the same value of a k-valued variable.

Definition 14. 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 sets $X=\{x_1,\ldots,x_n\}$ and $Y=\{y_1,\ldots,y_p\}$ specify their arguments. Let $X=\{X_1,..,X_s\}$ and $Y=\{Y_1,\ldots,Y_s\}$ be partitions of X and Y into s blocks and $Enc(X_1,Y_1),\ldots, Enc(X_s,Y_s)$ are encoding functions. Functions $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are called ***toggle equivalent under input encoding function*** $Enc(X,Y)=Enc(X_1,Y_1) \wedge \ldots \wedge Enc(X_s,Y_s)$ if $(\boldsymbol{f_1}(\boldsymbol{x}) \neq \boldsymbol{f_1}(\boldsymbol{x}') \wedge (Enc(\boldsymbol{x},\boldsymbol{y})=Enc(\boldsymbol{x}', \boldsymbol{y}')=1)) \Rightarrow (\boldsymbol{f_2}(\boldsymbol{y}) \neq \boldsymbol{f_2}(\boldsymbol{y}')$ and vice versa $(\boldsymbol{f_2}(\boldsymbol{y}) \neq \boldsymbol{f_2}(\boldsymbol{y}') \wedge (Enc(\boldsymbol{x},\boldsymbol{y})=Enc(\boldsymbol{x}', \boldsymbol{y}')=1)) \Rightarrow \boldsymbol{f_1}(\boldsymbol{x}) \neq \boldsymbol{f_1}(\boldsymbol{x}')$.

Proposition 6. Let $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ be toggle equivalent under input encoding function $Enc(X,Y)$=Enc$(X_1,Y_1) \wedge \ldots Enc(X_s,Y_s)$. Then $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ are "implementations" of the same multi-valued function of s multi-valued arguments.

Proof follows from Proposition 5 and Remark 4.

5. Common specification and toggle equivalence

In this section, we show that the existence of a CS of 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".

Definition 15. 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 16. Let N^* be a subcircuit of N. An input of a gate g of N^* is called ***an input*** of N^* if it is not connected to the output of some other gate of N^*. A gate g of N^* is called an ***internal gate*** if all the gates of N whose inputs are fed by the output of g are in N^*. Otherwise, g is called ***an external gate***. The output of an external gate is called ***an output*** of circuit N^*.

Definition 17. Let N^* be a subcircuit of N of k inputs and p outputs. Let N^+ be the circuit obtained from N by replacing N^* with a k-input, p-output node that inherits all the connections of subcircuit N^* to the gates of N that are not in N^*. We will say that N^+ is obtained from N by ***collapsing*** the subcircuit N^*.

Definition 18. Let $\{N^1, ...,N^k\}$ be a partition of N into subcircuits. This partition is called ***topological*** if for each pair of subcircuits N^i,N^j the following condition holds: if there is node g' of N_i and node g'' of N_j that are connected by a path in N and $level(g') < level(g'')$, then for any pair of nodes g^* and g^{**} (where g* is in N^i and g^{**} is in N^j) it is true that $level(g^*) < level(g^{**})$ ($level(g)$ is the level of g in N).

Definition 19. Let N be a Boolean circuit and $N^1, ..., N^k$ be a topological partition of N into subcircuits. Let T be a directed graph obtained from the DAG of N using the following steps:
1) Each subcircuit N^i is collapsed in N (i.e. replaced with a node G_i with the same number of inputs and outputs as N^i).
2) The outputs of each node G_i are merged into one. If two (or more) outputs of G_i are connected to inputs of node G_m, then all the inputs of G_m connected to G_i but one are removed.
T is called ***the communication specification*** corresponding to the topological partition $N^1, ..., N^k$.

Remark 5. Informally, T describes information flow between subcircuits $N^1, ..., N^k$. The output of G_i is connected to an input of G_k in T iff an output of N^i is connected to an input of N^k in N.

Remark 6. It is not hard to show that since $N^1, ..., N^k$ is a topological partition, T is a DAG (i.e. T is acyclic).

Definition 20. 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 longest path from an input of T to G_i is called the level of G_i and N^i (denoted by $level(G_i)$ and $level(N^i)$ respectively).

Definition 21. 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 $Cf(N_1^m,N_2^m)$, $m=1,\ldots,k$ the correlation functions computed exactly as it was described in Section 3. Namely, we first compute correlation functions for subcircuits of level 1, then for level 2 and so on. The correlation function $Cf(N_1^m,N_2^m)$ is obtained from the function $H=Sat(v(N_1^m)) \wedge Sat(v(N_2^m)) \wedge Cf^*$, the function Cf^* being the conjunction of correlation functions for all the subcircuits N_1^i,N_2^i whose outputs are connected to inputs of N_1^m , N_2^m. The function $Cf(N_1^m,N_2^m)$ is obtained from H by existentially quantifying away all the variables except the output variables of N_1^m, N_2^m.

Proposition 7. Let f_1 and f_2 be two Boolean functions that are toggle equivalent under input encoding function $Enc(X,Y)=Enc(X_1,Y_1) \wedge \ldots \wedge Enc(X_s,Y_s)$ (see Definition 14.) Let N_1 and N_2 be circuits implementing $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ and V and W be the output variables of N_1 and N_2. Let $Cf(V,W)$ be the function obtained from the function $Enc(X,Y) \wedge Sat(v(N_1)) \wedge Sat(v(N_2))$ by existentially quantifying away all the variables except variables of V and W. Then $Cf(V,W)$ is an encoding function.

Proof. Assume the contrary i.e. $Cf(V,W)$ is not an encoding function. Suppose, for example, that there are vectors $\boldsymbol{v},\boldsymbol{v}',\mathbf{w}$ such that $\boldsymbol{v} \neq \boldsymbol{v}'$ and $Cf(\boldsymbol{v},\boldsymbol{w})=Cf(\boldsymbol{v}',\boldsymbol{w})$. Since Cf is obtained by existentially quantifying away some variables, there must exist vectors $\boldsymbol{z}=(\mathbf{x},\boldsymbol{y},\ldots,\boldsymbol{v},\boldsymbol{w})$ and $\boldsymbol{z}' =(\boldsymbol{x}',\boldsymbol{y}',\ldots,\boldsymbol{v}',\boldsymbol{w})$ such that

1) input variables of N_1 and N_2 are assigned $\boldsymbol{x}$, $\boldsymbol{y}$ in $\boldsymbol{z}$ and $\boldsymbol{x}'$, $\boldsymbol{y}'$ in $\boldsymbol{z}'$ and

2) output variables of N_1 and N_2 are assigned $\boldsymbol{v},\boldsymbol{w}$ in $\boldsymbol{z}$ and $\boldsymbol{v}',\boldsymbol{w}'$ in $\boldsymbol{z}'$.

Since $\boldsymbol{v} \neq \boldsymbol{v}'$, then $\boldsymbol{x} \neq \boldsymbol{x}'$. Hence there are vectors $\boldsymbol{x},\boldsymbol{x}',\boldsymbol{y},\boldsymbol{y}'$ such that $\boldsymbol{x} \neq \boldsymbol{x}'$, $Enc(\boldsymbol{x},\boldsymbol{y})=Enc(\boldsymbol{x}', \boldsymbol{y}')=1$, $N_1(\boldsymbol{x})=\boldsymbol{v},N_1(\boldsymbol{x}')=\boldsymbol{v}'$ and $N_2(\boldsymbol{y})=\boldsymbol{w},N_2(\boldsymbol{y}')=\boldsymbol{w}$. But this means that N_1 and N_2 are toggle inequivalent and so we have a contradiction.

Proposition 8. Let $N_1^1, ..., N_1^k$ and $N_2^1, ..., N_2^k$ be topological partitions of functionally equivalent circuits N_1 and N_2 . Let communication specifications T_1 and T_2 of N_1 and N_2 with respect to these partitions be identical i.e. $T_1=T_2=T$. Let each pair of subcircuits N_1^m and N_2^m be toggle equivalent under input encoding function Cf^* (see Definition 21). Then N_1 and N_2 have a common specification S. The topology of S is specified by the communication specification T and N_1^m and N_2^m are implementations of m-th block of S.

Proof. By assumption each pair of circuits N_1^m,N_2^m are toggle equivalent if their inputs are restricted by the corresponding correlation functions. According to Proposition 6, if the correlation functions are encoding functions, then N_1^m and N_2^m implement the same multi-valued function. So one just needs to prove that all correlation functions are encoding functions. Let $level(N_1^m)=level(N_2^m)=1$. Then inputs of N_1^m and N_2^m are common primary inputs of N_1 and N_2 . One can view inputs of N_1^m and N_2^m as related by the encoding function that is the conjunction of functions describing equivalency of the corresponding input variables of N_1^m and N_2^m. According to Proposition 7, the correlation function relating outputs of N_1^m and N_2^m is an encoding function. Then by induction in topological levels one can easily prove that correlation function $Cf(N_1^p,N_2^p)$ is an encoding function for any value of p.

6. A method of HLLS

In previous sections we developed some theory that allows one to check the correctness of a CS of circuits N_1 and N_2 even if this specification is described implicitly. Suppose that N_1 and N_2 have the same topological specification defined by subcircuits N_1^1, ..., N_1^k and N_2^1, ..., N_2^k and corresponding pairs of subcircuits are toggle equivalent. Then these two sets of subcircuits give an implicit representation of a CS of N_1 and N_2. In this section, we use this result to formulate a method of High-Level structure aware Logic Synthesis (HLLS).

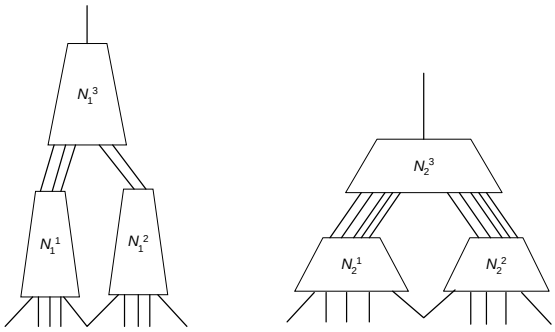

Figure 2. An example of HLLS

Let us describe this method by a simple example. Suppose circuit N_1 is partitioned into three subcircuits N_1^1, N_1^2, N_1^3 as shown in Figure 2 on the left. Suppose that we want to obtain a circuit N_2 with a better performance than N_1. This can be done by replacing "tall" subcircuits N_1^1, N_1^2, N_1^3 with "short" and "wide" subcircuits N_2^1, N_2^2, N_2^3 shown in Figure 2 on the right. In the method of HLLS this replacement is done in the following way. First we pick a subcircuit of topological level 1, say N_1^1 and synthesize a "short" subcircuit N_2^1 that is toggle equivalent to N_1^1. The same procedure applies to the other subcircuit of topological level 1. The subcircuit N_1^2 is replaced with a toggle equivalent subcircuit N_2^2. After we are done with level 1, we move to topological level 2 i.e. to the subcircuit N_1^3.

However, before the re-synthesis of circuit N_1^3 we need to compute the necessary correlation functions. The inputs of N_1^3 are connected to the outputs of circuits N_1^1 and N_1^2 , so we need to compute the correlation functions $Cf(N_1^1,N_1^2)$ and $Cf(N_1^2,N_2^2)$. This computation is done as described in Section 3. Then we build a subcircuit N_2^3 that is toggle equivalent to the subcircuit N_1^3. The toggle equivalence of N_1^3 and N_2^3 is computed not "globally" (in terms of primary inputs of N_1 and N_2) but locally in terms of inputs of N_1^3 and N_2^3 related by the correlation functions $Cf(N_1^1 , N_2^1)$ and $Cf(N_1^2,N_2^2)$. Since subcircuits N_1^3 and N_2^3 have only one output, their toggle equivalence means that N_1^3 and N_2^3 (and hence circuits N_1 and N_2) are functionally equivalent modulo negation. Since N_1 and N_2 have the same topological specification and the corresponding subcircuits are toggle equivalent, N_2 is a different implementation of the same three-block specification that is implemented by N_1.

Of course, the described method gives only "the big picture" because the procedure for synthesis of toggle equivalent circuits is not described. The generalization of the method to a circuit N_1 with a specification of an arbitrary number of subcircuits is straightforward. The subcircuits of a specification of N_1 are replaced with toggle equivalent subcircuits in topological order. Toggle equivalence of subcircuits N_1^m and N_2^m is computed locally in terms of inputs of N_1^m and N_2^m related by correlation functions obtained before.

7. Relation of HLLS to other synthesis procedures

In this section, we discuss the relation between HLLS and existing synthesis procedures based on local transformations. The main three differences between such procedures and HLLS are shown in Table 1. The key idea of HLLS is to restrict the search to implementations of a predefined specification. The justification of such an approach is as follows. Suppose we need to optimize a combinational circuit N_1 that has a high-level specification S represented as a partition of N_1 into subcircuits N_1^1, ..., N_1^k. One can view these subcircuits as implementations of multi-valued blocks G_1,.,G_k of S obtained by encoding values of variables of S with binary codes (however it does not mean that this encoding was performed explicitly).

It is quite possible that the chosen codes are far from optimal and so a much better implementation N_2 of S than

N_1 may exist. However, a procedure based on local transformations, be it don't care based optimization [5] or SPFDS [6][7], will not be able to find N_2. This is because to produce a different implementation of S one has to perturb many nodes of N_1 at once (when replacing a subcircuit N_1^m with its toggle equivalent counterpart). On the other hand, it is unlikely that by local transformations one can produce a circuit (that is not an implementation of S) "better" than N_2.

Table 1. Comparison of HLLS and procedures based on local transformations

Synthesis based on local transformations	*HLLS*
Circuit is optimized node by node	Many nodes are perturbed "at once"
Equivalence checking is not an issue	Reducing complexity of equivalence checking is crucial
The search space is not restricted	One considers only implementations of a predefined specification.

Of course, the conjecture that the quality of encodings matters so much is based on the assumption that the specification S captures essential features of the circuit to be implemented. An example, of a circuit with a "meaningful" specification is a multiplier. On the other hand, if N_1 has many specifications, sticking to one of them does not make much sense.

It is not hard to see that HLLS and synthesis procedures based on local transformations are complementary and so can be used together. An HLLS procedure can be used as a first step of logic optimization meant to improve encodings of multi-valued variables and so to generate a better starting point for a procedure based on local transformations.

8. A procedure for testing toggle equivalence

Checking toggle equivalence is a key operation of the method of HLLS described in Section 6. In this section, we describe a procedure for testing toggle equivalence and give some experimental results. Let N_1 and N_2 be two multi-output Boolean circuits. Denote by $inp(N_1)$,$inp(N_2)$ and $out(N_1)$,$out(N_2)$ the set of primary input and output variables of circuits N_1 and N_2. Denote by $Enc(inp(N_1),inp(N_2))$ a function specifying allowable combinations of inputs for N_1 and N_2. Let N^*_1 and N^*_2 be copies of circuits N_1 and N_2 respectively that depend on different sets of variables (shown in Fig. 2.) Denote by H the function $Sat(v(N_1)) \wedge Sat(v(N_2)) \wedge Sat(v(N^*_1)) \wedge Sat(v(N^*_2)) \wedge Enc(inp(N_1),inp(N_2)) \wedge Enc(inp(N^*_1),inp(N^*_2))$. Denote by $Eq(out(N_1),out(N^*_1))$ the function that is equal to 1 iff $out(N_1)$ and $out(N^*_1)$ take the same value. Denote by $Neq(out(N_1),out(N^*_1))$ the function that is equal to 1 iff $out(N_1)$ and $out(N^*_1)$ take different values. Denote by H_1 the function $H \wedge Eq(out(N_1),out(N^*_1)) \wedge Neq(out(N_2),out(N^*_2))$. Denote by H_2 the function $H \wedge Neq(out(N_1),out(N^*_1)) \wedge Eq(out(N_2),out(N^*_2))$.

Proposition 9. Circuits N_1 and N_2 are toggle equivalent iff the functions H_1 and H_2 (defined above) are constant 0.

If part. Assume the contrary i.e. H_1 and H_2 are constants 0 and N_1 and N_2 are not toggle equivalent. Suppose there are vectors $(\boldsymbol{x},\boldsymbol{h})$ and $(\boldsymbol{x}^*,\boldsymbol{h}^*)$ such that $N_1(\boldsymbol{x}) = N_1(\boldsymbol{x}^*)$ while $N_2(\boldsymbol{h}) \neq N_2(\boldsymbol{h}^*)$ where $Enc(\boldsymbol{x},\boldsymbol{h})=Enc(\boldsymbol{x}^*,\boldsymbol{h}^*)=1$. Then there is a vector $\boldsymbol{w}$ that satisfies the function H_1 (and so we have a contradiction). Indeed, let $\boldsymbol{w}$ be the set of assignments to the variables of circuits N_1,N_2,N_1^*,N^*_2 under the input assignment $(\boldsymbol{x},\boldsymbol{h},\boldsymbol{x}^*,\boldsymbol{h}^*)$. The assignment $\boldsymbol{w}$ satisfies $Eq(out(N_1),out(N^*_1))$ because $N_1(\boldsymbol{x}) = N_1(\boldsymbol{x}^*)$. It also satisfies $Neq(out(N_2),out(N^*_2))$, because $N_2(\boldsymbol{h}) \neq N_2(\boldsymbol{h}^*)$. Besides, $\boldsymbol{w}$ satisfies function H (because $Enc(\boldsymbol{x},\boldsymbol{h})=Enc(\boldsymbol{x}^*,\boldsymbol{h}^*)=1$ and $\boldsymbol{w}$ is picked in such a way that it satisfies $Sat(v(N_1))$, $Sat(v(N_2))$, $Sat(v(N^*_1))$, $Sat(v(N^*_2))$.

Only if part. Assume the contrary i.e. circuits N_1 and N_2 (and hence N^*_1 and N^*_2) are toggle equivalent but either H_1 or H_2 (or both) is satisfiable for some assignment of values. Suppose for example that there is an assignment $\boldsymbol{w}$ such that $H_1(\boldsymbol{w})=1$. Let $\boldsymbol{x}$, $\boldsymbol{h}$, $\boldsymbol{x}^*$,$\boldsymbol{h}^*$ be the parts of vector $\boldsymbol{w}$ that are assignments to the input variables of N_1,N_2,N^*_1,N^*_2 respectively. Since vector $\boldsymbol{w}$ satisfies $Sat(v(N_1))$, $Sat(v(N_2))$, $Sat(v(N^*_1))$, $Sat(v(N^*_2))$., then the rest of the assignments in $\boldsymbol{w}$ are consistent with $\boldsymbol{x}$,$\boldsymbol{h}$, $\boldsymbol{x}^*$,$\boldsymbol{h}^*$ i.e. they are the values of gates of N_1,N_2,N^*_1,N^*_2 under the input vector $\boldsymbol{x}$,$\boldsymbol{h}$,$\boldsymbol{x}^*$,$\boldsymbol{h}^*$ respectively. Besides, since $Enc(\boldsymbol{x},\boldsymbol{h})= Enc(\boldsymbol{x}^*,\boldsymbol{h}^*)=Eq(out(N_1),out(N^*_1))\wedge Neq(out(N_2),out(N^*_2))=1$, then $N_1(\boldsymbol{x}) = N_1(\boldsymbol{x}^*)$ while $N_2(\boldsymbol{h}) \neq N_2(\boldsymbol{h}^*)$. This means that N_1 and N_2 are not toggle equivalent and so we have a contradiction.

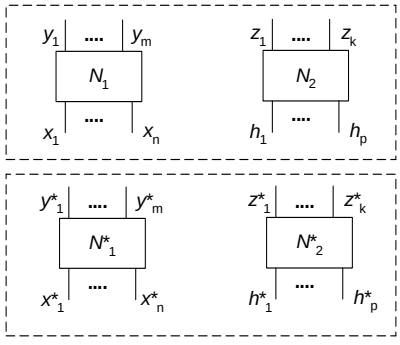

Figure 3 Two copies of circuits N_1 and N_2 to be checked for toggle equivalence

In the experiments we used 39 circuits from the MCNC benchmark set. Namely, for each circuit N_1 listed in Table 2 we checked its toggle equivalence with a circuit N_2 produced from N_1 by a random permutation of outputs. Clearly, permutation of outputs destroys functional equivalence of N_1 and N_2 but preserves their toggle equivalence. To check if N_1 and N_2 are toggle equivalent

Table 2. Toggle equivalence checking for MCNC circuits

name	#inputs	#outputs	time (sec.)
pcler8	27	17	0.01
frg1	28	3	0.03
sct	19	15	0.04
unreg	36	16	0.04
lal	26	19	0.05
c8	28	18	0.07
cht	47	36	0.08
b9	41	21	0.09
my_adder	33	17	0.16
example2	85	66	0.17
C432	36	7	0.18
apex7	49	37	0.18
vda	17	39	0.18
ttt2	24	21	0.33
i5	133	66	0.40
i6	138	67	0.63
term1	34	10	0.74
i7	199	67	0.99
i9	88	63	0.99
K2	45	43	1.46
apex6	135	99	1.58
x4	94	71	1.58
x3	135	99	1.70
x1	51	35	2.28
C499	41	32	2.41
rot	135	107	2.97
C880	60	26	5.07
frg2	143	139	5.13
C1355	41	32	6.48
pair	173	137	9.22
des	256	245	40.39
C1908	33	25	47.10
too_large	38	3	70.9
i8	133	81	150.02
C5315	178	123	193.10
C3540	50	22	261.28
dalu	75	16	310.67
i10	257	224	588.87
C7552	207	108	6,122.5
Geometric mean			1.84
Arithmetic mean			200.77

we created CNFs describing functions H_1 and H_2 and checked them for satisfiability. (Since tested pairs N_1,N_2 were toggle equivalent, all generated CNF formulas were unsatisfiable.) For satisfiability testing we used the SAT-solver BerkMin[1],[2]. The runtimes are shown in the last column of Table 2. It is not hard to see that in the majority of cases, toggle equivalence was established very quickly which proves that the proposed procedure may be used in HLLS.

9. Conclusions

We introduce a method of High-Level structure aware Logic Synthesis (HLLS). The key idea of the method is to re-encode multi-valued variables of the specification describing the high-level structure of the circuit implicitly at the gate level. We show that this can be done by preserving toggle equivalence among initial and re-synthesized implementations of multi-valued blocks. We show that toggle equivalence can be efficiently checked for relatively large pieces of combinational logic, which makes HLLS a promising direction for future research.

References

[1] BerkMin web page. *http://eigold.tripod.com/BerkMin.html*

[2] Goldberg E.,Novikov,Y. *BerkMin: A fast and robust SAT-solver*. Design, Automation, and Test in Europe (DATE '02), pages 142-149, March 2002.

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

[4] E.Goldberg, Y.Novikov. *How good can a resolution based SAT-solver be?* In the proceedings of 6-th International Conference on Theory and Applications of Satisfiability Testing, Italy, 2003,LNCS 2919,pp.37-52.

[5] H.Savoj. Don't cares in multi-level network optimization. Ph.D. thesis, UC Berkeley, 1992.

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

[7] S.Yamashita, H.Sawada, and A.Nagoya. A New method to express functional permissibilities for LUT based FPGAs and its applications. ICCAD, pp. 254-261, 1996.