Topologically Constrained Logic Synthesis

Subarnarekha Sinha
University of California at Berkeley
Berkeley, CA 94720

Alan Mishchenko
Portland State University
Portland, OR 97207

Robert K.Brayton
University of California at Berkeley
Berkeley, CA 94720

Abstract

SPFDs, a mechanism for expressing flexibility during logic synthesis, were first introduced for FPGA synthesis. They were then extended to general, combinational Boolean networks and later the concept of sequential SPFDs was introduced. In this paper, we explore the idea of using SPFDs for functional decomposition. A new type of functional decomposition called topologically constrained decomposition is introduced. An algorithm is provided for solving this problem using SPFDs. Preliminary experimental results are encouraging and indicate the feasibility of the approach. A scheme is also presented for generating instances of the topologically constrained decomposition problem.

1 Introduction

SPFDs or Sets of Pairs of Functions to be Distinguished provide a powerful new formalism for expressing the implementation flexibility of a node during logic synthesis. They were first introduced in the context of FPGA synthesis [1] but were shown to have the ability for expressing the flexibility of a node in general, combinational Boolean networks [2], as well as sequential circuits [3]. SPFDs are a type of Multiple Boolean Relation (MBR)[1] [4] and thus can express flexibility that cannot be expressed using don't cares or Boolean relations. In most previous applications of SPFDs, the flexibility expressed by them has been used for simplifying the nodes in a combinational network or for state re-encoding during sequential synthesis.

In this paper, we present a new application of SPFDs to a particular type of functional decomposition. This problem is motivated by the so-called wireplanning problem in which the wires or communication channels are planned before synthesis. In Section 2, we describe some previous work in functional decomposition. Section 3 provides some motivation for the relevance of SPFDs in functional decomposition. The topologically constrained decomposition problem is introduced in Section 4. Section 5 describes an algorithm for using SPFDs for solving this problem. The correctness of the procedure is also established here. Some preliminary experimental results are provided in Section 6. Section 7 provides some ideas for generating instances of the topologically constrained decomposition problem. The paper concludes with some directions for future work in Section 8.

2 Previous Work

Decomposition is a fundamental problem in logic synthesis. Its goal is to break a function into smaller functions. The problem can be stated as

$$F(X) = G(H(X_1), X_2),$$

[1] An MBR can express more flexibility than an SPFD.

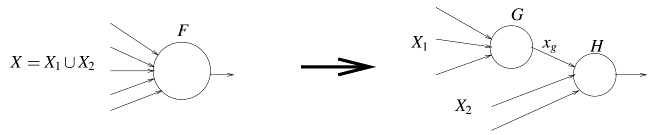

Figure 1: Ashenhurst Decomposition.

c \ ab	00	01	11	10
0	0	0	1	0
1	1	1	0	1

Figure 2: Decomposition Chart.

$$X_1 \cup X_2 = X.$$

Generally, G and H are less complex than F. It is known, in the worst case the circuit size realizing an n-input logic function is $O(2^n/n)$. If $F(X)$ has a decomposition $G(H(X_1), X_2)$, the worst case for the decomposed circuit is $O(2^{n_1}/n_1 + 2^{n_2+1}/(n_2+1))$, where $n_1 = |X_1|$ and $n_2 = |X_2|$. Thus functional decomposition can reduce the circuit size exponentially.

The first systematic study of decomposition [5] characterized the existence of a simple disjoint decomposition of a function. This is a special case of the above equations, where $X_1 \cap X_2 = \phi$ and G is a single output function. The problem is shown in Figure 1.

The set X_1 is called the bound set and X_2 the free set. We describe this procedure in some detail to explain the basic idea behind functional decomposition.

The necessary and sufficient condition for the existence of a decomposition was given in terms of the decomposition chart $D(X_1|X_2)$ for F for the partition $X_1|X_2$. A decomposition chart is a truth-table of F where the vertices of $B^n = \{0,1\}^n$ are arranged in a matrix. The columns of the matrix correspond to the vertices of $B^{X_1} = B^s$, and its rows correspond to the vertices of $B^{X_2} = B^{n-s}$. The entries in $D(X_1|X_2)$ are the values that F takes for all possible input combinations. For example, if $F(a,b,c) = ab\overline{c} + \overline{a}c + \overline{b}c$, the decomposition chart for F for the partition $ab|c$ is shown in Figure 2.

Ashenhurst proved the following result, which relates the existence of a decomposition to the number of distinct columns in the decomposition chart $D(X_1|X_2)$.

Theorem 1 *(Ashenhurst) A simple disjoint decomposition exists if and only if the corresponding decomposition chart has at most two distinct columns.*

Two vertices x_1 and x_2 in B^s are compatible if they have the same column patterns. For an incompletely specified function, a don't care entry '-' cannot cause two columns to be incompatible. Thus, two columns c_i and c_j are compatible if for each row k, either $c_i(k) = -$, or $c_j(k) = -$, or $c_i(k) = c_j(k)$. For a completely specified function, compatibility is an equivalence relation and the set of vertices that are mutually compatible form an equivalence class. Hence the column multiplicity of the decomposition chart is the number of equivalence classes. For incompletely specified functions, the compatibility relation is not an equivalence relation, i.e. there may be a case when $i \sim j \wedge j \sim k$, but $i \not\sim k$. So, a column may be contained in two or more compatible sets and a nontrivial procedure, like graph coloring, is needed for determining column multiplicity.

Since then, many more complicated functional decomposition models have been introduced that don't require either the bound set and the free set to be disjoint or the node G to have a single output. Recent research in the field also includes work on BDD-based methods aimed at improving the efficiency of decomposition [6, 7].

3 SPFDs and Decomposition

We briefly review SPFDs and their ability to represent the information content of a node (see [2] for a detailed overview) An SPFD, $R = \{(g_{1a}, g_{1b}), (g_{2a}, g_{2b}), \cdots, (g_{na}, g_{nb})\}$, denotes a set of pairs of functions that have to be distinguished i.e. for each pair $(g_{ia}, g_{ib}) \in R$, the minterms in g_{ia} has to produce a different value from the minterms in g_{ib}. The functions contained in an SPFD are all the functions that can satisfy the SPFD. A function f is said to satisfy an SPFD, $R = \{(g_{1a}, g_{1b}), (g_{2a}, g_{2b}), \cdots, (g_{na}, g_{nb})\}$, if for each pair $(g_{ia}, g_{ib}) \in R$, $f(g_{ia}) \neq f(g_{ib})$. An SPFD , $R = \{(g_{1a}, g_{1b}), \ldots, (g_{na}, g_{nb})\}$, can also be represented as a graph, $G = (V, E)$, where

$$
\begin{aligned}
V &= \{m_k | m_k \in g_{ij}, 1 \leq i \leq n, j = \{a, b\}\} \\
E &= \{(m_i, m_j) | ((m_i \in g_{pa}) \wedge (m_j \in g_{pb})) \vee \\
&\quad ((m_i \in g_{pb}) \wedge (m_j \in g_{pa})), 1 \leq p \leq n\}
\end{aligned}
$$

Every $e \in E$ is referred to as an SPFD edge. It is easy to see that any valid coloring of the SPFD graph is a multi-valued function that satisfies the SPFD.

An SPFD attached to a node specifies which pairs of primary input minterms can be or have to be distinguished by the node. This can be thought of as the information content of the node, since it tells what information the node contributes to its surrounding network.

Example 1 *Consider the simple node shown in Figure 3. Input A has the ability to distinguish* 00 *and* 01 *from* 10 *and* 11. *Similarly, input B has the ability to distinguish* 00 *and* 10 *from* 01 *and* 11. *Thus, the two inputs together can distinguish every input minterm from every other input minterm. However, the output of the node only has the ability for distinguishing* 00 *from* 10, 01 *and* 11.

A single-input single-output node (buffer or inverter) does not lose information. Any single-output node that depends on more than one

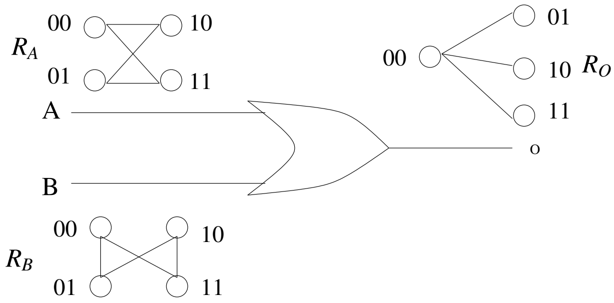

Figure 3: Information Flow through an OR-gate: R_O is a subset of $R_{IN} = R_A \cup R_B$

input always results in a loss of information. However, an n-input n-output node, whose function is reversible (i.e. for each input combination there is exactly one output combination, and vice versa) does not lose information.

Now consider the problem of disjoint decomposition. Consider the example shown in Figure 1 and look at it in terms of information flow. The original function F required that the onset minterms have to be distinguished from the offset minterms. Each input of F does a part of the distinguishing job. Now, if we want to re-implement F as the decomposed circuit shown in Figure 1, it is necessary that the new node G should be able to do all the distinguishing that the inputs in X_1 did for the function at F. In order to achieve this, let's look at the following algorithm, **com_decomp_w_spfd**.

1. *Compute the SPFD of F in terms of the input space $X = X_1 \cup X_2$. Denote it as R_F.*

2. *Remove all edges of R_F that can be distinguished by the inputs in X_2. Denote this new SPFD as R'.*

3. *Existentially quantify out the variables associated with the inputs in X_2 from R' to get the SPFD of the node, G. Denote this SPFD as R_G.*

The new function at G can be obtained by coloring R_G. Similarly, the new function at F can be obtained by expressing R_F in terms of $(x_g \cup X_2)$ and coloring it. It can be shown that Ashenhurst decompositions can be obtained using the above algorithm.

SPFDs can also be used for obtaining a non-disjoint decomposition. If a fanin, x_i, belongs only to X_2, then it has to be assigned to the SPFD of partition X_2. On the other hand, if a fanin, x_i, belongs to both X_1 and X_2, then we could choose to assign an edge distinguished by x_i to either the SPFD of partition X_1 or the SPFD of partition X_2. Assigning it to X_1 could increase the complexity of G whereas assigning it to X_2 could increase the complexity of H.

Given that SPFDs can be used for obtaining simple functional decompositions, an interesting decomposition scheme can be developed.

4 Topologically Constrained Decomposition Problem

A generalization of the decomposition idea to an arbitrary network of nodes is shown in Figure 4. Here, instead of specifying the free set and the bound set, the topology of the network is given i.e. the

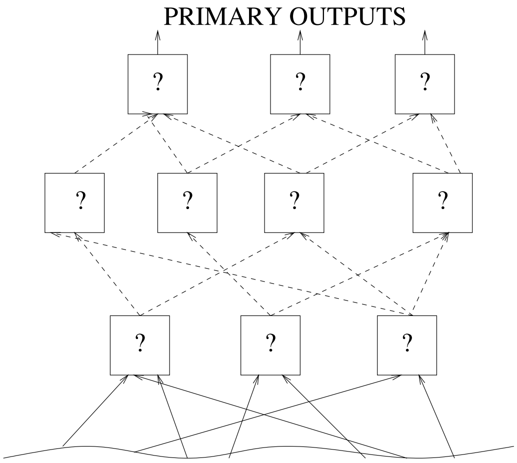

Figure 4: Problem Definition.

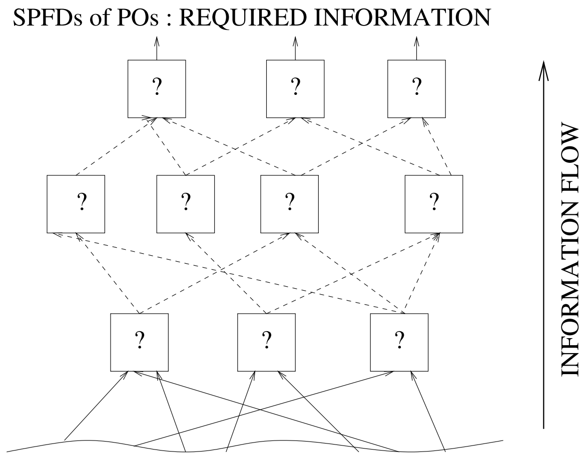

Figure 5: Information flows through the network.

fanin and fanout connections of all the nodes in the network are provided. The problem is to determine the functionalities of the nodes so that the network implements the required output functions. The nodes in the network can have multiple outputs (or equivalently can be multi-valued). The number of outputs of the nodes in the network are not specified a priori. However, it is desirable to have as many nodes with binary outputs as possible. The configuration could be generated by a wireplanning algorithm, where the communication between the boxes is specified but the actual contents of each box is not specified.

In the rest of this paper, we discuss the condition that the network topology has to satisfy in order to ensure that the network can be synthesized and a particular approach based on SPFDs for synthesizing the nodes.

5 Problem Solution

5.1 Preliminaries

Definition 1 *A* **cut** *is a set of nodes in the network that when removed completely isolates the primary inputs from the primary outputs.*

Obviously, a network can have many cuts.

Given a network, $\mathcal{N}$, let S denote its required input-output specification. The functions of the nodes in $\mathcal{N}$ are not known. Consider a node, η_j, in network, $\mathcal{N}$. Let L_j denote its level in the network, according to some topological order. A node, η_j, in $\mathcal{N}$ will be associated with two variables, y_j and y'_j. The variables associated with the primary inputs of $\mathcal{N}$ are collectively denoted as X or X', depending on whether the unprimed or primed variables are used. Both X and X' are referred to as the primary space. Similarly, the variables associated with the fanins of a node, η_j, are collectively denoted as Y_j or Y'_j. In the sequel, both Y_j and Y'_j are often referred to as the fanin space of η_j. The fanin and fanout nodes of η_j are collectively denoted as $FI(\eta_j)$ and $FO(\eta_j)$, respectively. Similarly, the transitive fanins and transitive fanouts of η_j are collectively denoted as $TFI(\eta_j)$ and $TFO(\eta_j)$, respectively. The primary outputs in the transitive fanout of η_j are denoted as $PO(\eta_j)$. R_j^{max} denotes the maximum SPFD of η_j. The SPFD of η_j that is used for deriving its new function is denoted as R_j. The synthesized function at η_j is denoted as f_j. R_j is expressed either in terms of the $(Y_j \cup Y'_j)$ space or the $(X \cup X')$ space. $\mathcal{G}(X, Y_j)$ is the characteristic relation connecting the primary inputs of $\mathcal{N}$ with the fanins of η_j.

5.2 Algorithm

In this algorithm, we use the analogy of information flow through the network. The network specification tells us what information needs to be passed on from the primary inputs to the primary outputs. As discussed, SPFDs can be used for denoting the information content of a node. So, the network function specification can be thought of as the information content of the primary outputs and can be re-expressed as SPFDs associated with the primary outputs. Similarly, the information content of the primary inputs can be expressed as SPFDs associated with the primary inputs. It is instructive to think of SPFDs of the primary outputs as the **required information** and the SPFDs of the primary inputs as the **available information** (as shown in Figure 5). The task of the synthesis process is to determine the information flow through the nodes in the network so that the required information is present at the primary outputs. A network can be thought of as a lossy information channel. For this method to work, it is necessary to ensure that the available information is not less than the required information. This translates into a topology constraint given in Lemma 1.

Lemma 1 *The network of empty nodes has to satisfy the following requirement : each primary output should have at least one path to each primary input in its true support.*

Note that the constraints given by Lemma 1 are only a necessary condition that each network topology has to satisfy. However, a two level network that expresses a primary output solely in terms of the primary inputs is also a network topology that satisfies Lemma 1. Hence, this lemma is not useful for generating any multi-level network topologies. In a later section, we will discuss some techniques that can be used for generating some interesting network topologies. The condition in Lemma 1, however, ensures that each edge in the

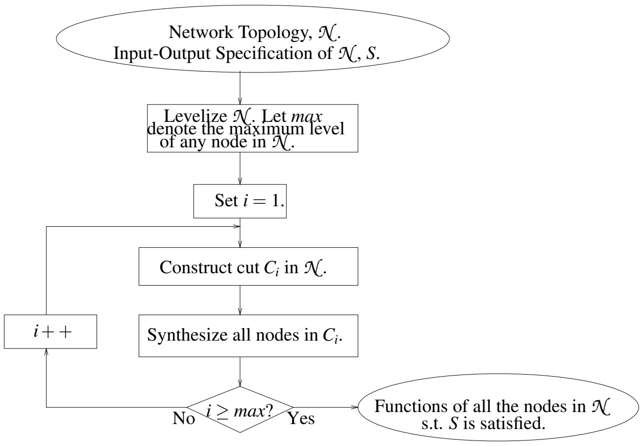

Figure 6: Algorithm.

SPFD of a primary output is contained in the SPFDs of one or more primary inputs in its transitive fanin. This way of framing the problem in terms of SPFDs enables us to utilize some of the familiar techniques of SPFD manipulation for determining the flow of information through the network from the primary inputs to the primary outputs.

Several other papers [8, 9] exploit the connection between information flow and synthesis. The first paper takes an evolutionary approach towards network synthesis, where a function is corrected by adding either a few constants or variables until it becomes the specified function. The second paper starts with a function expressed in terms of its primary inputs and progressively decomposes the function at each step until some user-defined limit like the number of fanins of each node is reached. This method looks at the information content of each node for determining the function of the fanins. At each node, either a serial or parallel decomposition is allowed. However, this method does not use a fixed network topology.

The basic idea of our algorithm is to ensure that the information necessary to meet the network specification is not lost as we move from the primary inputs to the primary outputs. It accomplishes this by defining a set of cuts in the network starting from the primary inputs and moving towards the primary outputs, and ensuring that each cut has all the necessary information. The general flow of the algorithm is shown in Figure 6.

There are two basic steps in the algorithm:

1. Defining the cuts in the network.
2. Computing the SPFDs of the nodes in the cut and synthesizing these nodes using their respective SPFDs.

Many schemes could be used for either of two steps; we describe one scheme for each step.

5.3 Defining the Cuts in the Network

The procedure goes as follows :

1. *Levelize all the nodes in the network starting from the primary inputs. For each primary input,* η_i, $L(\eta_i) = 0$. *For any other node,* $\eta_i \in \mathcal{N}$,

$$L(\eta_i) = max\{L(\eta_j) : (\eta_j \textit{ is a fanin of } \eta_i)\} + 1.$$

2. *Let* **max** *denote the maximum level of any node in the network. Define the cuts in the network starting from the primary inputs to the primary outputs. So, for* $i = \{0, 1, \cdots, \mathbf{max}\}$,

   (a) *For each primary output,* η_k, *compute* C_{ik} *to include*

      i. *All nodes in the transitive fanin of* η_k *with level* $= i$

      ii. *All nodes with level* $< i$ *that directly fanout to a node with level* $> i$ *in the transitive fanin of* η_k.

   Thus,

$$\begin{aligned} C_{ik} \;=\; & \{\eta_j | (\eta_j \in TFI(\eta_k)) \wedge (L(\eta_j) = i)\} \\ & \cup \{\eta_j | (L(\eta_j) < i) \wedge [\exists_{\eta_p} (L(\eta_p) > i) \\ & \wedge (\eta_p \in FO(\eta_j)) \wedge (\eta_p \in TFI(\eta_k))]\}. \end{aligned}$$

   (b) *Construct* $C_i = \cup_k C_{ik}$. *Thus,* C_i *includes all nodes in* $\mathcal{N}$ *with level* $= i$ *and all nodes with level* $< i$ *that directly fanout to a node with level* $> i$. *Note that* C_i *is a cut in* $\mathcal{N}$ *as removing these nodes will completely disconnect the primary inputs from the primary outputs.* C_{ik} *denotes the subset of nodes of* C_i *which provides all the information to a primary output,* η_k.

C_0 consists of the primary inputs of $\mathcal{N}$. A node η_j definitely appears in cut $C_{L(\eta_j)}$. Furthermore, let $lmax = max\{L(\eta_k) | \eta_k \in FO(\eta_j)\}$. Then, η_j also appears in a cut, C_i, where $L(\eta_j) < i < lmax$. Thus, two cuts in $\mathcal{N}$ can share some nodes.

5.4 Synthesizing the nodes in the cut

Here, we briefly describe the algorithm for synthesizing the nodes in a particular cut, C_i. The main requirement that has to be satisfied after the synthesis step is that for each primary output, η_k, C_{ik} must be able to provide all the information that η_k requires. In the rest of the section, we describe the algorithm that ensures that this condition is satisfied.

The cuts are synthesized from the primary inputs to the primary outputs. Hence, when the nodes in C_i are being synthesized, all the nodes in cuts $C_1, \cdots, C_{i-1}$ have already been synthesized. The nodes in C_i that have already been synthesized are denoted as C_i^r. These are the nodes of level$< i$. The nodes in C_i with level $= i$ have to be synthesized and are denoted as C_i^u. Note that $C_i = C_i^u \cup C_i^r$.

The algorithm **syn_cuts** first orders the nodes in C_i according to some heuristic such that all the nodes in C_i^r are earlier in the ordering than all the nodes in C_i^u. It then computes the maximum SPFD of each node in C_i^u. This maximum SPFD denotes the total set of edges that a node can distinguish, derived solely from the distinguishing ability of its fanins. The SPFD computation then proceeds from the nodes earlier in the ordering to the ones later in the ordering. At each node, η_j, its SPFD, R_j, is derived from its maximum SPFD as follows: For each primary output, $\eta_k \in PO(\eta_j)$, the algorithm determines the edges in $R_k(X, X')$ that cannot be distinguished by the remaining nodes in C_{ik} (C_{ik} is the subset of nodes of C_i that provide all the information to η_k). The node's SPFD , R_j, is simply the union of all these edges. The new function at η_j is derived from R_j. Then, the algorithm moves to the next node in the cut.

Algorithm **syn_cuts**($\mathcal{N}$, S):

1. *Assume that each primary output, η_k, has an SPFD, $R_k(X,X')$, associated with it. This can be derived from the given specification, S.*

2. *Order the nodes in C_i. All the nodes in C_i^r should be earlier in the ordering than all the nodes in C_i^u.*

3. *For each node, $\eta_j \in C_i^u$, compute the maximum SPFD of the node and denote it as R_j^{max}.*

$$R_j^{max}(X,X') = (\cup_{\eta_p \in FI(\eta_j)} R_p(X,X')),$$

where $R_p(X,X')$ is the SPFD of η_p expressed in terms of the primary input space[2]*. Thus, $R_j^{max}(X,X')$ denotes the maximum set of edges that η_j can distinguish. However, if we just assign all the edges in R_j^{max} to R_j, a lot of information will be duplicated in the network. Hence, we try to minimize the amount of redundant information in R_j in the next step.*

4. *Process the nodes in C_i^u in order, starting from the one earliest in the ordering. For each node, $\eta_j \in C_i^u$*

   (a) *For each $\eta_k \in PO(\eta_j)$,*

      i. *Determine the edges in the SPFD of η_k that can only be distinguished by η_j according to the ordering computed in Step 2. Hence, from*

$$R_{jk}(X,X') = R_j^{max}(X,X') \wedge R_k(X,X'),$$

      A. *Remove the edges that* **are** *distinguished by the SPFDs of the nodes in C_{ik} that are earlier in the ordering. Thus, for each $\eta_n < \eta_j$,*

$$R_{jk}(X,X') \leftarrow R_{jk}(X,X') \wedge \overline{R_n(X,X')}.$$

      B. *Remove the edges that* **can be** *distinguished by the nodes in C_{ik} that are later in the ordering. Thus, for each $\eta_m > \eta_j$,*

$$R_{jk}(X,X') \leftarrow R_{jk}(X,X') \wedge \overline{R_m^{max}(X,X')}.$$

   (b) *Compute $R_j(X,X') = \cup_{k=1}^{n} R_{jk}(X,X')$, where $n = |PO(\eta_j)|$.*

   (c) *Compute*

$$R_j(Y_j,Y_j') = \exists_{X,X'} \mathcal{G}(X,Y_j)\mathcal{G}(X',Y_j')R_j(X,X').$$

   This is the image of $R_j(X,X')$ to the local input space of η_j.

   (d) *Determine the new function at η_j by coloring $R_j(Y_j,Y_j')$ and minimizing the resulting ISF using ESPRESSO-MV. Let this new function be f_j. Note that f_j can be multi-valued, in general.*

5. *Stop.*

[2]Since η_p is a fanin of η_j, hence it has already been synthesized and has an SPFD associated with it.

5.4.1 Global SPFDs vs Local SPFDs

In all our computations, the SPFDs were expressed in terms of the primary inputs (global SPFDs) instead of the local inputs (local SPFDs). While computations of global SPFDs can be fairly memory intensive, the disadvantages of expressing the SPFDs of the nodes in terms of the local fanin space are two- fold:

1. Expressing the SPFD in terms of the local space can add some extra useless edges. For instance, suppose the primary input edge (x_1,x_2) produces the edge (y_1,y_2) in the local fanin space of η_j. Now, if we take the inverse image of (y_1,y_2) back to the primary input space, then in addition to (x_1,x_2), a few more edges may be obtained. Hence, expressing R_j^{max} in terms of the local inputs could add some useless edges. This, in turn, may result in some useless edges in the SPFD, R_j, that is used for deriving the new function at η_j.

2. Translating the SPFD from one local space to another also results in some loss of precision due to early existential quantification. Thus, suppose we want to remove the edges in the SPFD, R_p, of η_p from the SPFD, R_j^{max}. The current algorithm would do the following (as shown in Step 4(a)(A)):

$$R_j(Y_j,Y_j') = \exists_{X,X'}(R_j^{max}(X,X')\overline{R_p(X,X')})\mathcal{G}(X,Y_j)\mathcal{G}(X,Y_j').$$

   On the other hand, if all the SPFDs were expressed in terms of the local fanin spaces, the computation would be the following:

$$\begin{aligned} R_j(Y_j,Y_j') &= R_j^{max}(Y_j,Y_j') \wedge (\exists_{Y_p,Y_p'}\overline{R_p(Y_p,Y_p')} \\ &\quad En(Y_j,Y_p)En(Y_j',Y_p')), \end{aligned}$$

   where $En(Y_j,Y_p) = \exists_X \mathcal{G}(X,Y_j)\mathcal{G}(X,Y_p)$. So, in the second equation, the quantification is done first, followed by the conjunction. This could result in some additional edges.

In practice, these disadvantages were indeed operative. Hence, all our computations are performed on global SPFDs.

5.5 Correctness

Lemma 2 *Let C_{ik}^r and C_{ik}^u denote the synthesized and unsynthesized nodes of C_{ik}, respectively. Then,*

$$C_{(i-1)k} = C_{ik}^r \cup C^{add},$$

where $C^{add} = \{\eta_p | (\eta_p \in FI(\eta_j)) \wedge (\eta_j \in C_{ik}^u)\}$.

Proof Sketch:

$\rightarrow : C_{(i-1)k} \subseteq C_{ik}^r \cup C^{add}$:

Consider a node, $\eta_p \in C_{(i-1)k}$. Either η_p fans out to at least one node of level $> i$ or else the maximum level of its fanouts is $= i$. In the first case, it belongs to C_{ik}^r. In the second case, it belongs to the C^{add}.

$\leftarrow : C_{ik}^r \cup C^{add} \subseteq C_{(i-1)k}$:

Any node $\eta_p \in C_{ik}^r$ has level $< i$ and fans out to at least one node of level $> i$. Thus, $\eta_p \in C_{(i-1)k}$. Consider a node, $\eta_p \in C^{add}$. All nodes in C_{ik}^u have level $= i$. Thus, $L(\eta_p) \leq (i-1)$. Two cases must be distinguished.

1. $L(\eta_p) = (i-1)$: Since η_p is in the transitive fanin of η_k, hence $\eta_p \in C_{(i-1)k}$.

2. $L(\eta_p) < (i-1)$: Since, η_p has at least one fanout of level $= i$, thus it fans out to at least one node with level $> (i-1)$. Hence, $\eta_p \in C_{(i-1)k}$. □

Theorem 2 *If the topology constraint given by Lemma 1 is satisfied, each primary output, η_k, can always be synthesized to satisfy its network specification. The internal nodes in the network can be multi-valued after synthesis.*

Proof Sketch:
We prove the above for an arbitrary primary output, η_k.
Base case: The topology constraint ensures that C_{0k} has all the information that η_k requires.
Inductive step: Suppose C_{ik} has all the information required by η_k. We prove that the algorithm **syn_cuts** ensures that $C_{(i+1)k}$ will have all the information that η_k requires.

Suppose that's not true. Then, there exists an edge $e = (x,x') \in R_k(X,X')$ that cannot be distinguished after synthesizing the nodes in $C_{(i+1)k}$. This happens only if e is not in the SPFDs of the nodes in $C^r_{(i+1)k}$ nor does it appear in the union of the maximum SPFD of the nodes in $C^u_{(i+1)k}$. Note that all nodes in $C^u_{(i+1)k}$ have level $= (i+1)$. Since the maximum SPFD of a node is simply the union of the SPFDs of its fanin nodes, e does not belong to the SPFDs of any of the fanin nodes of $C^u_{(i+1)k}$. But the fanins of the nodes in $C^u_{(i+1)k}$ together with the nodes in $C^r_{(i+1)k}$ form the nodes in C_{ik} (Lemma 2). Thus e does not belong to the SPFDs of the nodes in C_{ik}. But this contradicts our assumption that e can be distinguished by the nodes in C_{ik}. □

In the sequel, we refer to the entire algorithm shown in Figure 6 (comprised of defining the cuts in a network and synthesizing the nodes in each cut using **syn_cuts**) as **syn_spfd**.

6 Experiments

In this section, we describe some experiments that we performed for determining the practical feasibility of our scheme. As mentioned before, our algorithm needs a network topology and an input-output specification as its starting point. We are still currently investigating different techniques for generating network topologies and suggest one in Section 7. For now, we use circuits from the ISCAS benchmark suite (or their derivatives) and use their topology information and input-output specifications as our starting point. The experiments are set up to test if our procedures are valid and if they can closely reproduce the original circuit.

In the first set of experiments, the initial topology of a given ISCAS benchmark circuit and its input-output specification is used as the starting point. Thus, given the topology of the original circuit, **syn_spfd** is used for synthesizing the nodes in these networks. The initial results are shown in Table 1. Columns 2 and 3 show the literal counts of the original circuit and the circuit after using **syn_spfd**. An average improvement of 6.56% in literal count is obtained after using **syn_spfd**. One negative artifact of the greedy edge distribution scheme used in **syn_cuts** is that some of the nodes may be multi-valued [3]. This is because a node that appears later in the ordering in a cut may have to distinguish many edges and its SPFD graph may no longer be bipartite. Practical results, however, indicate that on average only 6.39% of the nodes are multi-valued (Column 4).

[3]Note that a solution exists for the given starting topology and the input-output specification in which all the nodes are binary. This solution is the original circuit.

circuits	original	**syn_spfd**	% MV-nodes
apex7	292	291	9.09
cht	236	199	0
cmb	62	77	2.22
cc	99	102	0
cu	90	88	0
f51m	195	194	0
lal	224	226	8.92
ttt2	339	298	8.51
term1	625	343	24.07
x2	71	59	11.11
Average	0	-6.56	6.39

Table 1: Results of using **syn_spfd** on ISCAS benchmark circuits

circuits	script.rugged	**syn_spfd**	% MV-nodes	simplify
apex7	246	261	3.03	255
cht	165	196	9.75	189
cmb	51	59	16.67	53
cc	63	64	0	64
cu	60	62	0	61
f51m	119	124	8.33	123
lal	106	107	0	107
ttt2	219	287	23.4	254
term1	176	169	6.67	160
x2	48	49	0	49
Average	0	7.97	6.78	3.87

Table 2: Results of using **syn_spfd** on optimized ISCAS benchmark circuits

Table 2 provides results of using our scheme on optimized ISCAS benchmark circuits. In this experiment, a circuit is optimized using *script.rugged* and the topology of the optimized circuit is used as the starting point of **syn_spfd**. The input-output specification is the functionality of the original circuit. This experiment is set up to test if our procedure works under tighter topology constraints. The results indicate we can reproduce the optimized circuit quite closely. Even though, we do worse than the optimized circuit in some cases, the average increase in literal count in only about 8%. The average percentage of nodes that are multi-valued is 6.78%. The results of Table 2 should be viewed from the point of view of a real application. In a real application, we would not know a solution. We would be given only the topology and the input-output specifications of the network. There would be no way for judging the quality of our solution. Table 2 supports the claim that the solution is pretty good. In addition, network optimization methods which do not alter the topology like don't care optimization can be applied for improving the solution. The results of running **full_simplify** [11] is shown in Column 5 in Table 2. Thus essentially we are able to almost recover the heavily optimized circuit that we started with.

Our preliminary results indicate that it is possible to use the algorithm described in this paper for synthesizing the nodes in a network from its topology and its input-output specification. In our experiments with the topologies of the unoptimized circuits, there

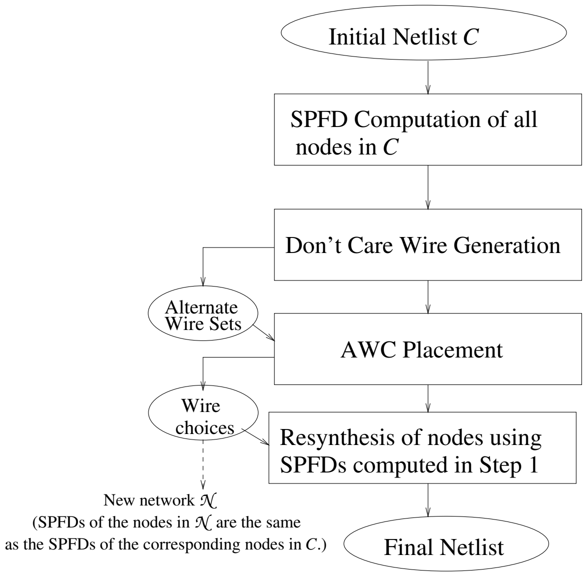

Figure 7: Don't Care Wires.

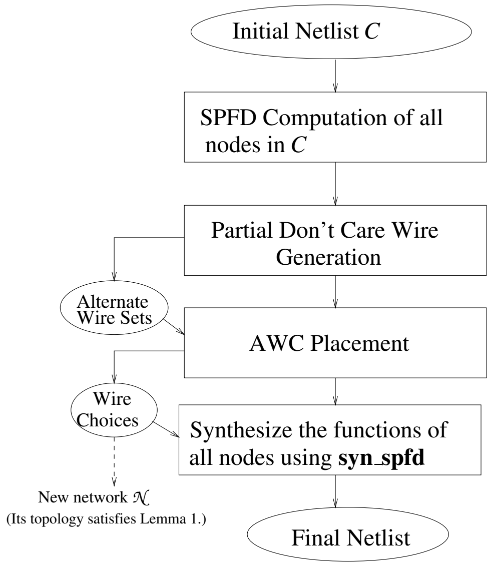

Figure 8: Partial Don't Care Wires.

was also a reduction in the literal count with respect to the original circuit. Also, in practice, we found that on average less than 7% of the nodes are multi-valued in both the optimized and unoptimized topologies.

The work up to this point assumes that a network topology is provided to us. In the next section, we briefly discuss a technique for generating a network topology that can be used as an input to our algorithm, **syn_spfd**.

7 Generating Instances of Topologically Constrained Decomposition Problem

In the previous sections, we provided an algorithm for solving the topologically constrained decomposition problem. It was assumed that the data for the problem was given. Here, we briefly describe a scheme for generating instances of the problem which can then be input to the algorithm **syn_spfd**.

Our approach is a generalization of the use of *don't care* wires [12]. A very brief overview of that approach and its benefits are provided.

The idea was to compute a set of alternate wires for each wire in the original circuit C. In addition, the choice of an alternate wire for one wire had to be independent of the choices made for the other wires. This enabled a placement algorithm to pick alternates for all the wires in C at the same time, thereby obtaining a more globally optimized solution. Below, we give a brief description of the algorithm used in [12] for generating these "compatible" sets of alternate wires.

Let S_{kj} denote the set of alternate wires for wire $w_{\eta_k \to \eta_j}$.

Definition 2 *Given a set of sets of nodes* $S = \{S_{kj}\}$*, a* **selection** *is a ordered set of nodes* $\{\eta_1, \ldots, \eta_{|S_{kj}|}\}$ *such that* $\eta_k \in S_{kj}$.

Definition 3 *A set of sets* $S = \{S_{kj}\}$ *is* **compatible** *if for each selection, there exist logic functions at each node such that the implied netlist for that selection can implement the specifications at the primary outputs.*

The basic idea for generating the sets S_{kj} is the following: Given a circuit C, the SPFDs of all the nodes are computed using the algorithm presented in [2]. Then the sets S_{kj} are computed such that the following two properties are satisfied:

1. For any selection of $\{S_{kj}\}$, the resulting network is acyclic.

2. Given S_{kj}, $w_{\eta_s \to \eta_j} \in S_{kj} \longrightarrow R_{w_{\eta_k \to \eta_j}} \subseteq R_s$, where $R_{w_{\eta_k \to \eta_j}}$ and R_s denote the SPFDs of $w_{\eta_k \to \eta_j}$ and η_s, respectively.

It was shown that any set of sets S_{kj} that satisfied the above two properties is compatible. Given the sets of alternate wires, a modified placement algorithm called AWC was used for selecting an alternate wire for each wire such that the total wirelength (i.e. the sum of the half perimeters of the bounding boxes of each net) was minimized. The choice of an alternate wire for each wire in C created a new network $\mathcal{N}$ with an improved topology from the point of view of total wirelength. The SPFDs of the nodes in $\mathcal{N}$ were the same as the SPFDs of the same nodes in C because the alternate wires were selected such that all the information that was originally available at any node in C was still available at the corresponding node in $\mathcal{N}$. However, the local functions of the nodes in $\mathcal{N}$ were no longer the same as the corresponding ones in C. Hence a new function had to be synthesized at each node using its SPFDs for ensuring that $\mathcal{N}$ implemented the same function as C. The entire flow is shown in Figure 7.

It was found that 7.5% of the wires had alternate wire sets and an improvement of 12% in wirelength was obtained after using these sets. Experimental results also indicated a positive correlation between the improvement in wirelength and the percentage of wires that had alternate wires. The generalization that we present in the rest of the section is an attempt at increasing the number of wires that have alternate wires.

In [12], wire $w_{\eta_s \to \eta_j}$ is considered an **alternate** for wire $w_{\eta_k \to \eta_j}$ only if the SPFD of $w_{\eta_k \to \eta_j}$ is completely contained in the SPFD of η_s (Condition 2 above). In practical experiments, it was found that this requirement might be too restrictive, thereby limiting the number of wires that have alternate wire sets. Here we relax the requirement for a wire to be an alternate for another wire in an attempt to increase the number of wires with alternate wire sets.

We say that wire $w_{\eta_s \to \eta_j}$ is an **alternate** for wire $w_{\eta_k \to \eta_j}$ if the following condition is satisfied:

$$\mathcal{X}(w_{\eta_k \to \eta_j}) \subseteq \mathcal{X}(\eta_s),$$

where $\mathcal{X}(w_{\eta_k \to \eta_j}) = \{\eta_i | \ (\eta_i \text{ is a primary input }) \ \wedge (\exists_{(x_1,x_2) \in R_{w_{\eta_k \to \eta_j}}} (x_1(i) \neq x_2(i)))\}$ and $\mathcal{X}(\eta_s) = \{\eta_i | \ (\eta_i \text{ is a primary input }) \ \wedge (\exists_{(x_1,x_2) \in R_s} \ (x_1(i) \neq x_2(i)))\}$. The terms $x_1(i)$ and $x_2(i)$ denote the value of η_i in the minterms x_1 and x_2, respectively. The $\mathcal{X}$ values for all the nodes and wires in C are computed in a topological fashion from the primary inputs to the primary outputs.

The above condition checks if the set of all primary inputs that can distinguish all the edges in the SPFD of $w_{\eta_k \to \eta_j}$ is contained in the set of all primary inputs that can distinguish all the edges in the SPFD of η_s. This ensures that all the primary inputs that are needed for distinguishing the edges in $R_{w_{\eta_k \to \eta_j}}$ are definitely contained in the transitive fanins of η_s, independent of the wiring changes in the transitive fanin of η_s. However, this condition does not try to match the edges in $R_{w_{\eta_k \to \eta_j}}$ and R_s as the original condition of an alternate. Hence, this requirement on an alternate is more relaxed than the requirement on an alternate in [12]. We believe that by using it more wires in C will have alternate wire sets. We call these sets of alternate wires *partial don't care* wires because now an alternate wire may only provide a fraction of the information that the original wire provides. Note that during the construction of the sets S_{kj} in our scheme, we use the same restrictions that were used in [12] for ensuring that any selection of S_{kj}, the resulting network is acyclic.

The flow shown in Figure 7 can be modified to obtain an instance of the topologically constrained decomposition problem. The new flow is shown in Figure 8. Using this definition of alternates, the sets S_{kj} are constructed. Then the AWC placement algorithm is used for selecting an alternate for each wire such that the total wirelength is minimized. The network $\mathcal{N}$ that is obtained after this selection process has a topology that satisfies Lemma 1. This is because this definition of alternates ensures that the set of all primary inputs that are needed to distinguish all the edges in the SPFD of a primary output (i.e. the primary inputs in its true support) in C have a path to it. Hence, $\mathcal{N}$ is an instance of topologically constrained decomposition problem. The algorithm **syn_spfd** can then be used to synthesize the functions of the nodes in $\mathcal{N}$ so that it implements the same function as the initial netlist C. Note that since an alternate of a wire may now provide only a part of the information provided by the original wire, the SPFDs of the nodes in $\mathcal{N}$ are not the same as the SPFDs of the corresponding nodes in C. Hence, the simple resynthesis algorithm used in Figure 7 cannot be used.

8 Conclusions and Future Work

This paper discusses the synthesis process for "topologically constrained decomposition". The initial results are very encouraging. Using our algorithm, we were able to recover the starting circuit to a great extent using just its topology information and its input-output specification. One problem was the presence of some multi-valued nodes in a topology that has an implementation where all the nodes are binary. Since the multi-valued nodes had to be encoded for determining the final literal count of the synthesized circuits, they also contribute to a significant fraction of the total literal count. We are currently investigating schemes for reducing the number of multi-valued nodes in the synthesized circuits. We plan to experiment with some ordering schemes to offset the negative effects of the greedy edge distribution scheme. The current ordering scheme uses a random ordering for the unresynthesized nodes in a cut. In addition, we are also investigating schemes for edge distribution that attempt to limit the chromatic number of the SPFDs of the nodes in the cut. Another problem of the present algorithm is the memory usage of the global SPFDs. There are a few approaches for dealing with this problem. It may be beneficial from the memory perspective to represent the global SPFDs as asymmetric relations. Another help in this direction is the use of SAT as described in [10]. Also, for large networks, this algorithm can be applied to portions of a partitioned network.

We are currently investigating schemes for generating an initial topology. One idea involves using a generalization of the concept of "don't care" wires as discussed in Section 7. We also believe this algorithm could be useful in other wireplanning scenarios where the interconnect structure is planned out before the node functionalities are decided.

References

[1] S. Yamashita, H. Sawada, and A. Nagoya. A new method to express functional permissibilities for LUT based FPGAs and its applications. In *International Conference on Computer-Aided Design*, 1996.

[2] S. Sinha and R.K. Brayton. Implementation and Use of SPFDs in Optimizing Boolean Networks. In *International Conference on Computer-Aided Design*, Nov 1998.

[3] S. Sinha, A. Kuehlmann and R. Brayton. Sequential SPFDs. In *International Conference on Computer-Aided Design*, Nov 2001.

[4] E. M. Sentovich and R. K. Brayton. Multiple Boolean Relations. In *International Workshop on Logic Synthesis*, May 1993.

[5] R.L. Ashenhurst. The Decomposition of Switching Functions. In *International Symposium on Theory of Switching Functions*, 1959.

[6] M. Perkowski and S. Grygiel. A survey of literature on Functional Decomposition. *Technical Report, Department of Electrical Engineering, Portland State University*, 1995.

[7] S. Hassoun, T. Sasao and R. Brayton. Logic Synthesis and Verification. *Kluwer Academic Publishers*, 2002.

[8] V. Cheushev, S. Yanushkevich, V. Shmerko, C. Muraga and J. Kolodziejcyk. Information Theory Method for Flexible Network Synthesis. In *IEEE International Symposium on Multiple-Valued Logic*, May 2001.

[9] L. Jozwiak. Information Relationships and Measures in Application to Logic Design. In *IEEE International Symposium on Multiple-Valued Logic*, May 1999.

[10] S. Sinha and R. Brayton. Improved Robust SPFD Computations. In *Internatinational Workshop on Logic Synthesis*, Jun 2001.

[11] A. Mishchenko and R.K. Brayton. Simplification of Non-Deterministic Multi-Valued Networks. Submitted to *Internatinational Workshop on Logic Synthesis*, Jun 2002.

[12] P. Chong, Y. Jiang, S. Khatri, F. Mo, S. Sinha and R. Brayton. Don't Care Wires in Logical/Physical Design. In *International Workshop on Logic Synthesis*, Jun 2000.