Sequential SPFDs

Subarnarekha Sinha
University of California at Berkeley
Berkeley, CA 94720

Andreas Kuehlmann
Cadence Berkeley Labs
Berkeley, CA 94704

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

Abstract

SPFDs are a mechanism to express flexibility in Boolean networks. Introduced by Yamashita et al. in the context of FPGA synthesis [4], they were extended later to general combinational networks [2]. We introduce the concept of sequential SPFDs and provide an algorithm to compute them based on a partition of the state bits. The SPFDs of each component in the partition are used to generate equivalence classes of states. We provide a formal relation between the resulting state classification and the equivalence classes produced by classical state minimization of completely specified machines [6]. The SPFDs associated with the state bits can be applied for re-encoding the state space. For this, we give an algorithm to re-synthesize the sequential circuit using sequential SPFDs and the new state re-encoding.

1 Introduction

During *logic synthesis*, the implementation flexibility of a network is used to optimize it. Classically, the flexibility of the individual network nodes is represented using *Incompletely Specified Functions* (ISFs). *Don't Cares* are applied to extract this flexibility. These are of two forms: Satisfiability Don't Cares (SDCs) and Observability Don't Cares (ODCs) [1]. SDCs represent input combinations at a node that cannot be generated by the driving logic. ODCs are input combinations for which the output of the node is not observable at any primary output. ODCs are expensive to compute and need to be updated after every change of another node function. To avoid this, Compatible Observability Don't Cares (CODCs) [5] were introduced, which represent subsets of ODCs. They are efficiently computed by propagating CODCs in reverse topological order from the outputs to the inputs, and are independent; i.e., a CODC at one node can be used for optimizing the node function without affecting the validity of the CODCs at other nodes.

Yamashita et al. developed a new method to express flexibility, called *Sets of pairs of functions to be distinguished* (SPFD), in the context of FPGA synthesis [4]. These were extended [2] for general combinational networks. SPFDs represent a set of ISFs, in contrast to don't cares which express a single ISF. The SPFD of a node specifies which pairs of minterms have to be distinguished by the node function to ensure that enough information is available to resynthesize the fanout network. Since an SPFD specifies what the node has to distinguish, it can be considered the "information content" assigned to a node.

The classical computation of equivalence classes of states for finite state machines was introduced in [11]. There has been significant research in the area of finite state machine minimization (c.f. [6]). Most of these approaches suffer from state space explosion. In addition, the benefits of state minimization do not necessarily translate favorably to the final implementation of the sequential circuit. Approaches based on structural techniques try to solve this problem by working directly on the sequential circuit. The circuit structure is used to extract the set of unreachable states which are later applied as don't cares for circuit optimization ([7, 8]). However, these approaches still have to represent the entire state space and can potentially run into the state space explosion problem. To cope with this, local transformation techniques such as ATPG-based methods [9] and retiming and resynthesis [10] have been used. These are currently the most widely employed techniques, but do not capture the entire freedom of sequential optimization. Our approach avoids the state space explosion problem by using a partition of the state space while exploring more sequential freedom.

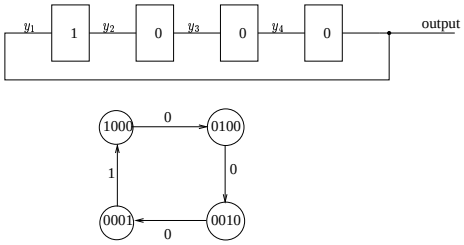

Figure 1: Example.

In [2], a procedure for computing compatible SPFDs for all nodes in a combinational network and a procedure to use them in resynthesis are presented. A more recent publication [3] gives more robust techniques for computing SPFDs. In this paper, we introduce the concept of sequential SPFDs. In Section 2, we illustrate how they can be used to reduce the number of state bits. Section 3 gives a procedure for computing sequential SPFDs and a proof that the union of the sequential SPFDs of the state bits contains the classical state incompatibility graph of the sequential circuit [11]. In Section 4, we give a procedure for resynthesizing a new circuit directly from the original one using sequential SPFDs.

2 Motivating Example

Example 1 *Figure 1 gives an example of a sequential circuit and its corresponding state transition graph (STG). It consists of four latches connected in series forming a shift register. The output of the fourth register is the only primary output of the circuit. The initial value of the first register is 1, the others 0.*

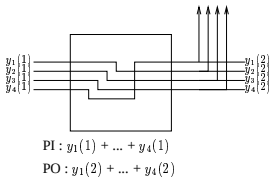

Figure 2: Combinational Circuit.

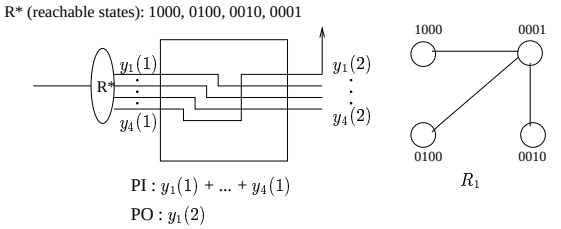

Figure 3: Modified Combinational Circuit.

This circuit is sequentially redundant and could be implemented with two registers only.

Combinational optimization treats the register inputs as primary outputs and the register outputs as primary inputs, and optimizes the combinational network between these boundaries. For the example in Figure 1, the resulting combinational network is shown in Figure 2. Obviously, combinational optimization techniques based on CODCs or SPFDs will not produce any circuit reduction.

Another way to apply combinational optimization techniques to a sequential circuit is to consider the primary inputs plus register outputs of the sequential circuit as primary inputs, and only the primary outputs of the sequential circuit as primary outputs. In addition, the register outputs can be constrained to allow combinations that correspond to states that are reachable. Thus, the example of Figure 1 yields the combinational optimization problem shown in Figure 3. Applying SPFDs on this combinational circuit, the SPFD of the primary output, which is the same as for $y_1(2)$, requires that all minterms that produce a 1 have to be distinguished from all the minterms that produce a 0. Its SPFD in terms of the present state bits $(y_1(1), y_2(1), y_3(1), y_4(1))$, denoted R_1, is shown in Figure 3: the minterm (0001) must be distinguished from the minterms (1000), (0100) and (0010). The SPFDs of the remaining state bits are empty. Thus, the union of the SPFDs of all state bits yields R_1. These are exactly the state pairs that produce different outputs in one transition. Thus SPFDs can provide information about the transitions of a sequential circuit, but it is not sufficient to just capture the information about one time frame. Informally speaking, it is necessary to unroll the circuit multiple times and determine the complete SPFDs at each node in a sequential circuit by computing the union of the SPFDs of the node in all time frames. We call these *sequential SPFDs*.

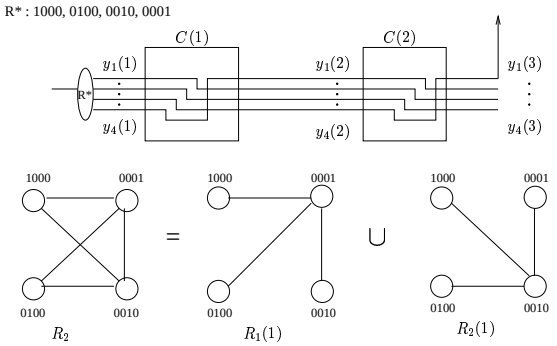

Figure 4: SPFDs obtained after unrolling once.

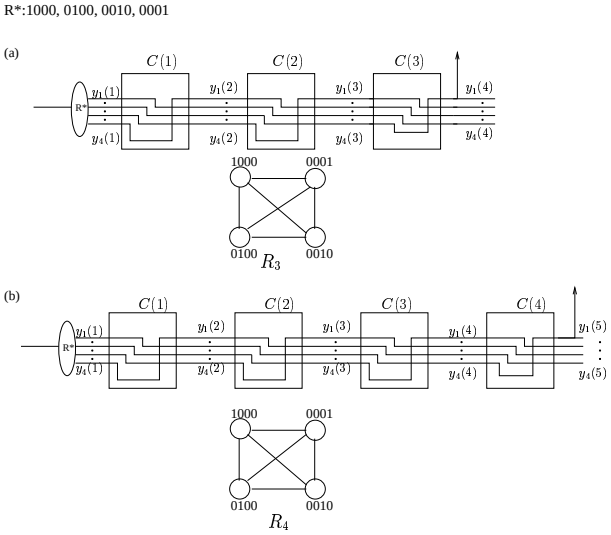

Figure 5: Various levels of unrolling and the corresponding SPFDs.

2.1 Sequential SPFDs

Consider a single unrolling of the circuit in Figure 1 which yields the combinational optimization problem shown in Figure 4. Denote the first and second copies by $C(1)$ and $C(2)$, respectively. The resulting combinational circuit has one primary output, $y_1(3)$, and four primary inputs, $y_1(1)$, $y_2(1)$, $y_3(1)$ and $y_4(1)$.

Computing the SPFDs of the nodes in the circuit and expressing the union of the SPFDs of the present state bits of $C(2)$ and $C(1)$ in terms of the present state bits of $C(2)$ and $C(1)$, respectively yields $R_1(1)$ and $R_2(1)$, shown in Figure 4. $R_1(1)$ is exactly the same as R_1 in Figure 3. $R_2(1)$ denotes the state pairs that produce different outputs after exactly two transitions. Hence, the union of the two SPFDs, $R_2 = R_2(1) + R_1(1)$, gives all those state pairs that produce different outputs in one or two transitions.

Unrolling the circuit once more and computing the SPFDs of all three copies gives R_3 shown in Figure 5(a). It has an edge between any two states that can produce different outputs

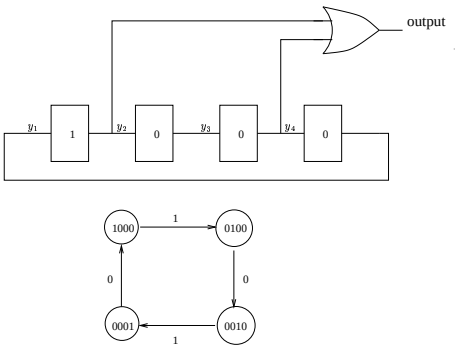

Figure 6: Another example.

in one, two, or three transitions.

Unrolling the circuit any further produces no additional edges. Thus R_3 includes all pairs of states that must be distinguished. If a pair of states s and s' has no edge between them, then the sequential circuit behaves identically, irrespective of whether it starts from s or s'. Hence, these two states could be merged. The graph R_3 can be colored to obtain equivalence classes for the states. Four colors are needed and hence two state bits are required to implement the circuit.

This example illustrates how progressive unrolling adds edges between state pairs (s, s') that behave differently in the future. In this particular example, since all states behave differently, we don't gain any additional information over the fact that the set of reachable states has four states and can be colored with four colors.

The next example illustrates how sequential SPFDs can provide useful information that cannot be gained just by examining the set of reachable states.

Example 2 *Consider the circuit in Figure 6. It is similar to that in Figure 1 except that the primary output of the circuit is now the OR of the first and third register outputs. The results for no unrollings and one unrolling are shown in Figure 7 and are denoted R_1 and R_2 respectively. R_1 and R_2 denote the state pairs that produce different outputs in one and two transitions respectively. Here, $R_1 = R_2$. We can stop the unrolling process since unrolling the circuit any further does not produce any more state pairs that behave differently in the future. Since R_1 is bipartite, only one state bit is required to implement the circuit.*

Thus, SPFDs can give useful relations between states which can be exploited for deriving a new state encoding. In the following section, we give a general procedure which uses SPFDs for re-encoding the state space. We also prove the correctness of the procedure.

3 Sequential SPFD Computation

3.1 Notation

For a sequential circuit M, denote the set of states by S, the set of transitions by T, the present state bits by Y, and next state bits by Y'. Let $y_i \in Y$ denote a present state bit of M and $y_i' \in Y'$ denote the next state bit corresponding to y_i. Let $\mathcal{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2, \cdots, \mathcal{Y}_k\}$, denote a partition of Y, where each

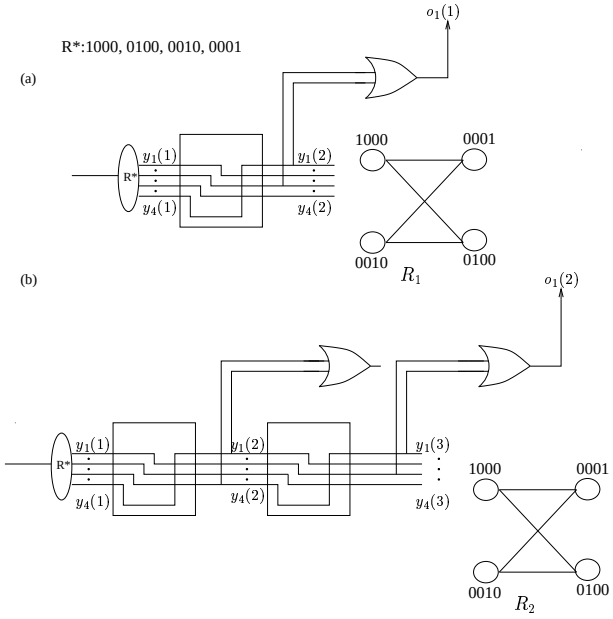

Figure 7: R_1 and R_2 .

$\mathcal{Y}_i$ is an individual component of $\mathcal{Y}$. Each node $\eta_j \in M$ has a sequential SPFD, R_j, associated with it.

Let C be the combinational circuit obtained from M where its primary inputs are the primary inputs plus the present state inputs of M, and primary outputs are the primary outputs and the next state outputs of M.

3.2 Algorithm

Algorithm **Com_seq_spfds** starts with the combinational circuit C obtained from M without unrolling. It computes the SPFDs of all nodes in C and uses them to update the sequential SPFDs of the nodes in M, which are initially empty. The SPFDs associated with the present state bits denote the information that they have to provide for ensuring correct functionality after one time frame. Next, the SPFD of each present state bit, y_i, is attached to the primary output of C that corresponds to y_i'. Then, the SPFDs of C are re-computed. In general, the i^{th} step computes the sequential SPFDs of the nodes required for correctness in $\leq i$ time frames. The process stops when no more edges are added to any node in the network.

Algorithm **Com_seq_spfds**(M, $\mathcal{Y}$):

1. $R^* =$ **Reach_states**(M).
2. *For each node $\eta_j \in M$, $R_j \leftarrow \phi$.*
3. *Obtain the combinational circuit C from M.*
4. *Restrict the present state inputs of C so that it allows only R^*; these are applied to restrict the number of input combinations that can be used during the image computation steps. The initial SPFDs on the POs of C that are also POs in M are given by the functions of the gates driving these outputs. The SPFDs of the remaining POs of C (i.e., the next state bits of M) are empty.*

5. **Compute_spfds**(C).
6. **Update_spfds**(M).
7. *repeat* {
   (a) **Modify_state_spfds**$(M, \mathcal{Y})$.
   (b) *Attach empty SPFDs to the POs of C that are also POs of M and the SPFDs of the present state bits of M to the POs of C that correspond to the next state bits of M.*
   (c) **Compute_spfds**(C).
   (d) **Update_spfds**(M).

   } *until (no change in SPFDs of nodes).*
8. *Stop.*

Reach_states computes the set of reachable states of M starting from the initial states. In general, any over-approximation of the reachable states can be used. However, the SPFDs of the nodes are the smallest if the exact set of reachable states is applied. **Compute_spfds** computes the SPFDs of all the nodes in C as in the combinational case([2, 3]). It proceeds from primary outputs to primary inputs and consists of two steps applied at each node in a reverse topological order; the SPFDs are first mapped from the node's local output space to its local input space and then distributed among its fanins. Subroutine **Update_spfds** uses the SPFDs of the nodes in C for updating the sequential SPFDs of the corresponding nodes of M. For each node η_j in M, it computes the union of the sequential SPFD of η_j stored in M with the new SPFD attached to the copy of η_j in C. During each SPFD computation phase, the present state bits are treated as primary inputs; hence the SPFD of each present state bit, y_i, is expressed in terms of the fanins of the fanouts of y_i. **Modify_state_spfds** transforms this SPFD so that it is solely expressed in terms of the state bits in $\mathcal{Y}_k$, where $y_i \in \mathcal{Y}_k$. The set of reachable states, R^*, is used to restrict the minterm combinations in the SPFD of y_i. It only contains edges between nodes a and b such that a and b are cubes of $\mathcal{Y}_k$ variables and are contained in S_k, where S_k is obtained from R^* by existentially quantifying the variables not in $\mathcal{Y}_k$.

For now, the algorithm assumes that $\mathcal{Y}$ has been chosen. It is important to observe that the algorithm only uses $\mathcal{Y}$ in the subroutine **Modify_state_spfds**. $\mathcal{Y}$ is useful if we want to do partial re-encoding of the state space since each component of the partition can be re-encoded independently. This avoids building an incompatibility graph over the entire state space. One simple heuristic to choose $\mathcal{Y}$ could group present state bits that have paths to the same set of primary outputs. In general, the structure of the circuit's topology can be exploited to find a good partition.

3.3 Theory

We formalize the ideas presented above. In general, M is a Mealy machine; its primary output logic is a function of the present state and the primary inputs.

Definition 1 *A pair of states (s, s') in S is* distinguishable *if there exists an input sequence such that M produces different output sequences for s and s'.*

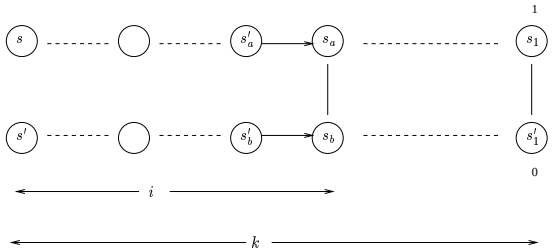

Figure 8: Illustration of the proof of Lemma 3

Definition 2 *Given a state s, the* projection *of s onto the set of variables Z, is obtained by existential quantification of all variables not in Z from s. This is denoted as s^Z.*

Definition 3 *The* sequential SPFD *at a node, η_j, is the SPFD, R_j, associated with it when* **Com_seq_spfds** *terminates.*

In the sequel, R_j^m denotes the SPFD of η_j after m steps of **Com_seq_spfds**.

Definition 4 *The* SPFD of a component, *$\mathcal{Y}_k \in \mathcal{Y}$ is the union of the SPFDs of the present state bits in $\mathcal{Y}_k$. It is denoted as $R_{\mathcal{Y}_k}$.*

Definition 5 *The* state SPFD, *R, is a graph $G = (S, E)$, where an edge exists between two states s and s' if there exists a component, $\mathcal{Y}_i \in \mathcal{Y}$, such that $(s^{\mathcal{Y}_i}, s'^{\mathcal{Y}_i}) \in R_{\mathcal{Y}_i}$. Here, $s^{\mathcal{Y}_i}$ and $s'^{\mathcal{Y}_i}$ are projections of s and s' respectively onto the state bits of $\mathcal{Y}_i$.*

We first show that the algorithm **Com_seq_spfds** terminates and then that the state SPFD, R, has an edge between a pair of states if they are distinguishable.

Lemma 1 *The computation of R_j^k of a node, η_j, by* **Com_seq_spfds** *is monotonic in k.*

Proof:
Let the SPFD of η_j after k and $(k+1)$ iterations be denoted as R_j^k and R_j^{k+1} respectively. Since R_j^{k+1} is obtained from R_j^k by adding SPFDs edges, hence $R_j^k \subseteq R_j^{k+1}$. □

Lemma 2 *R_j^k is finite for $k > 0$.*

Proof:
Let the inputs of η_j be denoted as Y_j. R_j^k denotes the input combinations that have to be distinguished after k iterations. Since η_j has a finite number of inputs, $R_j^k \subseteq Y_j \times Y_j$ is finite. □

Lemma 3 *If two reachable states s and s' are distinguishable, then the state SPFD, R, contains an edge between them.*

Proof:
By contradiction. Suppose s and s' are distinguishable in k steps but $(s, s') \notin R$. Then, there must be a set of states $\{s_a, s_b, s_a', s_b'\} \in S$ such that $(s_a', s_a) \in T$, $(s_b', s_b) \in T$, $(s_a, s_b) \in R$ and $(s_a', s_b') \notin R$. This is illustrated in Figure 8.

Suppose the algorithm stops after m steps. The stopping criterion requires that no more additional SPFD edges are added. Since $(s_a, s_b) \in R$, then $e = (s_a^{\mathcal{Y}_k}, s_b^{\mathcal{Y}_k})$ must exist in the

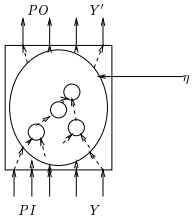

Figure 9: M': Implementing the Transition relation of M

SPFD of at least one component, $\mathcal{Y}_k$. This implies that there exists a present state bit, $y_j \in \mathcal{Y}_k$, s.t. its SPFD, R_j, contains e. Since $e \in R_j^m$, the algorithm would have added $e' = (s_a'^{\mathcal{Y}_l}, s_b'^{\mathcal{Y}_l})$ to the SPFD of a present state bit, y_i, in the next iteration, where $y_i \in \mathcal{Y}_l$. Hence, $e' \in R_{\mathcal{Y}_l}$. This contradicts our assumption $(s_a', s_b') \notin R$. □

Theorem 1 *The sequential SPFDs computed by* **Com_seq_spfds** *contains the information for correct re-encoding of a sequential machine.*

Proof:
R_j^k is monotonic (Lemma 1) and finite (Lemma 2) for all $k > 0$. Thus, R_j^k has a fixed point and hence **Com_seq_spfds** terminates. By Lemma 3, an edge exists in the state SPFD between any two reachable distinguishable states. □

3.4 Previous Work

The work presented in this paper is similar to classical state minimization of completely specified machines (c.f. [6]), which progressively partitions the state space into equivalence classes until no additional refinements can be made. At this point, the states in an equivalence class can be merged. Thus, each equivalence class contains states which are not distinguishable. By Lemma 3, the state SPFD contains an edge between any two states that are distinguishable. Hence, the states that can be colored with the same color are a subset of an equivalence class obtained by the classical state minimization. However, two states in the same equivalence class can have an edge between them in the state SPFD, since in general, only containment is guaranteed.

Consider the circuit M', shown in Figure 9. M' has a single multi-valued node, η, which implements the transition and output relations of M. The inputs of η are the primary inputs and the present state variables of M. The outputs of η are the primary outputs and next state variables of M. The state SPFD obtained by executing **Com_seq_spfds** on M' with $\mathcal{Y} = \{Y\}$ has an edge between two states iff they are distinguishable. In this case, the equivalence classes obtained from the state SPFD coincide with the ones obtained by the classical state minimization algorithm. Thus, the additional edges are due to the particular decomposition of M and the partitioning of the state bits.

3.5 State Encoding Using Sequential SPFDs

The SPFD of each component in the partition can be used to perform a re-encoding of the state space. For each component $\mathcal{Y}_i$, its SPFD, $R_{\mathcal{Y}_i}$, is solely expressed in terms of the variables of $\mathcal{Y}_i$. $R_{\mathcal{Y}_i}$ can thus be colored to get a new encoding of the bits of $\mathcal{Y}_i$. This procedure can be repeated for each $\mathcal{Y}_i$.

This method can accomplish a wide range of state encodings depending on the partition used while computing the SPFDs. On one extreme, if $\mathcal{Y} = \{Y\}$, then the SPFD of that component is equal to the state SPFD. Coloring the state SPFD yields a complete re-encoding of the state space. On the other extreme, re-encoding using $\mathcal{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2, \cdots, \mathcal{Y}_m\}$, where $\mathcal{Y}_i = y_i$, yields the original state encoding. A good partition that uses the initial decomposition of the circuit can be applied to do partial re-encoding of the state space. This approach is computationally feasible for very large machines since it only encodes a subset of the state variables in each step. Traditional state minimization algorithms must build an incompatibility graph over the entire state space.

The following example illustrates the effect of the different partitions of Y on the quality of state re-encoding.

Example 3 *Consider the circuit in Figure 6 and perform re-encoding of the state space for different partitions of Y.*

1. $\mathcal{Y} = \{\mathcal{Y}_1\}$, *where* $\mathcal{Y}_1 = (y_1, y_2, y_3, y_4)$. *After the first step, the SPFDs of y_2 and y_4 are obtained. The SPFDs of y_2 and y_4 are* $\{(0100, 1000), (0100, 0010)\}$ *and* $\{(0001, 1000), (0001, 0010)\}$, *respectively. Similarly, after the second step, the SPFDs of y_1 and y_3 are* $\{(1000, 0100), (1000, 0001)\}$ *and* $\{(0010, 0100), (0010, 0001)\}$. *Another step of the algorithm adds no more edges and thus the algorithm stops. The SPFD of $\mathcal{Y}_1$ is bipartite. Hence, this SPFD can be colored using two colors*[1]. *As a result, the reached states of the original state space can be encoded as:*

$$1000 \rightarrow 0; 0010 \rightarrow 0; 0100 \rightarrow 1; 0001 \rightarrow 1;$$

2. $\mathcal{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2\}$, *where* $\mathcal{Y}_1 = (y_1, y_3)$ *and* $\mathcal{Y}_2 = (y_2, y_4)$. *After the first step, the SPFDs of y_2 and y_4 are obtained. The SPFDs of y_2 and y_4 in terms of the variables in their respective components are* $\{(10, 00)\}$ *and* $\{(01, 00)\}$, *respectively. Similarly, after the second step, the SPFDs of y_1 and y_3 in terms of variables in their respective components are* $\{(10, 00)\}$ *and* $\{(01, 00)\}$. *The algorithm terminates in the next step. Consider the effect of re-encoding each partition separately. The SPFD of $\mathcal{Y}_1$ is* $\{(10, 00), (01, 00)\}$. *Since, it is bipartite, it can be colored using two colors. Let minterms* 00, 01 *and* 10 *in $\mathcal{Y}_1$ map to* 1, 0 *and* 0 *respectively. Similarly, $\mathcal{Y}_2$ can be re-encoded by coloring $R_{\mathcal{Y}_2}$. Since $R_{\mathcal{Y}_2}$ is also bipartite, it can also be colored with two colors. Let minterms* 00, 01 *and* 10 *in $\mathcal{Y}_2$ map to* 0, 1 *and* 1 *respectively. Hence, a circuit with two state bits can be obtained. So, the new encoding of the reached states is:*

$$1000 \rightarrow 00; 0010 \rightarrow 00; 0100 \rightarrow 11; 0001 \rightarrow 11;$$

3. $\mathcal{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3, \mathcal{Y}_4\}$, *where* $\mathcal{Y}_1 = y_1$, $\mathcal{Y}_2 = y_2$, $\mathcal{Y}_3 = y_3$ *and* $\mathcal{Y}_4 = y_4$. *The algorithm terminates in three steps and computes the SPFDs of all the nodes. The SPFDs*

[1] This is exactly what we got at the end of Section 2

of each y_i in terms of $\mathcal{Y}^{y_i}$ is $\{(1,0)\}$. Re-encoding each partition separately produces no reduction in the state bits.

3.6 Sequential SPFDs Based on the Classical Incompatibility Graph

It is interesting to note that the incompatibility graph of M derived using the classical state minimization algorithms ([6]) can be directly used to derive the sequential SPFDs of all the nodes of M in one step. The procedure is outlined below:

1. *Treat M as a specialized combinational circuit C where the POs are the POs of M and the PIs are the PIs of M plus the state bits of M. The next state bits of M are the inputs of a dummy node D in C. The SPFD of D is the supplied incompatibility graph.*
2. *Compute the SPFDs of all nodes in C (including D) in reverse topological order from primary outputs to primary inputs using* **Compute_spfds**.

4 Resynthesis Procedure

Given the encoding relation between the old states and the new states, Enc and the sequential SPFDs at all the nodes in M, the original circuit can be resynthesized using the following algorithm.

Algorithm **Seq_resyn**:

1. *Proceed in topological fashion from primary inputs and present state bits to primary outputs and present state outputs.*
2. *For each node, η_j, perform the following two steps:*
   (a) *Compute the mapping between the original and the new fanin spaces of node η_j:*

$$En(Y_j, \hat{Y}_j) = \exists_{X,Y,Y^e} R^*(Y)(Y^e = Enc(Y)) \; G(X,Y,Y_j)\hat{G}(X,Y^e,\hat{Y}_j),$$

   where X is the set of primary inputs, Y is the set of old state variables, Y^e is the set of new state variables, $R^*(Y)$ is the set of reachable states, Enc gives the new encoding of the states, $G(X,Y,Y_j)$ gives the transition relation of the original fanins and $\hat{G}(X,Y^e,\hat{Y}_j)$ the transition relation of the new fanins. The process is illustrated in Figure 10.
   (b) *Obtain the modified SPFD as*

$$R_j^{new}(\hat{Y}_j,\hat{Y}_j') = \exists_{Y_j,Y_j'} En(Y_j,\hat{Y}_j)En(Y_j',\hat{Y}_j') \; R_j(Y_j,Y_j').$$

   Color it to get an ISF for the node and minimize it.
3. *Attach a new multi-output node, F, at the output of the next state bits. F has n inputs and m outputs, where n is the number of original state bits and m is the number of new state bits. This node maps the re-implemented next state bits, $\hat{Y}'$, onto their new state encoding, $Enc(Y')$, as shown in Figure 11.*

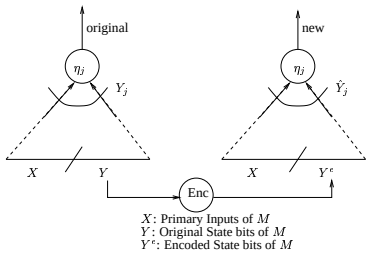

Figure 10: Deriving the encoding relation between the original and new fanin variables, $En(Y_j, \hat{Y}_j)$.

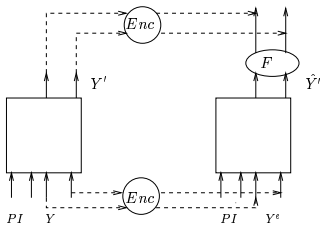

Figure 11: Computing the node of the multivalued function.

Example 4 *Figure 1 can be redrawn as shown in Figure 12. Assume that the sequential SPFDs of all the nodes are given. Further, let the encoding between the old and the new state spaces be $\{(1000,00),\ (0100,01),\ (0010,10),\ (0001,11)\}$. The sequential SPFD of f_1 is $\{(1000,0100),\ (1000,0010),\ (1000,0001)\}$. In terms of its inputs, the SPFD can be rewritten as $\{(0001,1000),\ (0001,0100),\ (0001,0010)\}$. The SPFD of f_1 in terms of its new fanins y_1^e and y_2^e is $\{(11,00),\ (11,10),\ (11,01)\}$. Thus, we can re-implement f_1 as $\hat{f}_1 = (\overline{y_1^e} + \overline{y_2^e})$. Similarly, the new functions of f_2, f_3 and f_4 are $\hat{f}_2 = \overline{y_1^e}\,\overline{y_2^e}$, $\hat{f}_3 = (y_1^e + \overline{y_2^e})$ and $\hat{f}_4 = y_1^e\overline{y_2^e}$ respectively. The encoding between Y' and $\hat{Y}'$ is*

$$\begin{aligned} 1000 &\rightarrow 0010 \\ 0100 &\rightarrow 1110 \\ 0010 &\rightarrow 1000 \\ 0001 &\rightarrow 1011. \end{aligned}$$

The new function of the output node F is given in Table 1. It can be implemented as two binary nodes, n_1 and n_2.

$$\begin{aligned} n_1 &= \hat{f}_1\overline{\hat{f}_2}(\hat{f}_3 \overline{\oplus} \hat{f}_4) \\ n_2 &= \hat{f}_1\hat{f}_3(\hat{f}_2 \oplus \hat{f}_4) \end{aligned}$$

The SPFD of the output in terms of its inputs is $\{(0001,1000),\ (0001,0100),\ (0001,0010)\}$. Given the new

$\hat{f}_1$	$\hat{f}_2$	$\hat{f}_3$	$\hat{f}_4$	$\hat{y}_1'$	$\hat{y}_2'$
0	0	1	0	0	0
1	1	1	0	0	1
1	0	0	0	1	0
1	0	1	1	1	1

Table 1: Function Table of MV-node

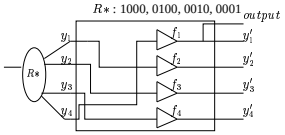

Figure 12: Revisiting the first example.

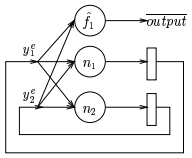

Figure 13: Re-implementation of Example 1.

state encoding, the modified SPFD is $\{(11,00),\ (11,01),\ (11,10)\}$. *Thus the new function of the output is* $y_1^e y_2^e$.

The above circuit can be further simplified by collapsing $\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$ *and* $\hat{f}_4$ *into* n_1 *and* n_2 *to yield the circuit shown in Figure 13.*

Similarly, resynthesizing the circuit in Figure 5 using the state encoding

$$1000 \rightarrow 1; 0100 \rightarrow 0; 0010 \rightarrow 1; 0001 \rightarrow 0;$$

and simplifying it yields the circuit in Figure 14. Note that in general **Com_seq_spfds** followed by **Seq_resyn** may be iterated to yield further reductions.

5 Conclusions and Future Work

Given a partition of the state bits, an algorithm was presented which computes sequential SPFDs for the nodes in a sequential circuit. Each component in the partition is also associated with an SPFD. The SPFDs of these components can be applied for re-encoding the state space. This approach can be particularly useful for larger machines as it avoids building the incompatibility graph for the entire state space. The effect of different partitions on the quality of results was illustrated. In the future, we plan to investigate different partitioning heuristics to come up with good criteria for partitioning the state space.

Another algorithm uses the sequential SPFDs and a new state encoding to resynthesize the sequential circuit. The resynthesis procedure can also be used in conjuction with other state minimization algorithms for obtaining a new circuit. The two algorithms can be iterated to yield new partitions and new encodings. The algorithms work directly on the current implementation of the machine and thus only deal with completely specified machines. A natural extension is to investigate the application of these ideas to incompletely specified machines.

Acknowledgements

This research was supported by the SRC under contract 683-002 and the California Micro program and our sponsors under this program, Fujistu, Cadence and Synopsys.

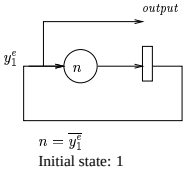

Figure 14: Re-implementation of Example 2.

References

[1] G.D. Hachtel and F. Somenzi. Logic Synthesis and verification algorithms. *Boston : Kluwer Academic Publishers*, 1996.

[2] S. Sinha and R. Brayton, "Implementation and use of SPFDs in optimizing boolean networks," in *ICCAD*, pages 103–110, 1998.

[3] S. Sinha and R. Brayton, "Improved Robust SPFD Computations", in *IWLS*,2001.

[4] S. Yamashita, H. Sawada, and A. Nagoya. A New Method to Express Functional Permissibilities for LUT based FPGAs and Its Applications. In *ICCAD*, pages 254–261, 1996.

[5] H. Savoj. Don't Cares in Multi-Level Network Optimization. *Ph.D. thesis, UC Berkeley*, 1992.

[6] T. Kam. T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Synthesis of finite state machines: logical optimization. *Boston : Kluwer Academic Publishers*, 1997.

[7] B. Lin, H. Touati and R. Newton. Don't Care Minimization of Multi-level Sequential Logic Networks. In *ICCAD*, pages 414–417, 1990.

[8] C. Lin, K. Chen, M. Marek-Sadowska and M. Lee. Sequential Permissible Functions and their Application to Circuit Optimization. In European Design and Test Conference, pages 334–339, 1996.

[9] K.T. Cheng and L.A. Entrena. Sequential logic optimization by redundancy addition and removal. In *ICCAD*, pages 310–315, 1993.

[10] S. Malik, E.M. Sentovich and R.K. Brayton. Retiming and Resynthesis: Optimizing sequential networks with combinational networks. In *IEEE Trans on Computer-Aided Design*, pages 74–84, 1991.

[11] Z. Kohavi. Switching and Finite Automata Theory. *McGraw Hill Publishing Company*, 1978.