SHAPE-BASED SEQUENTIAL MACHINE ANALYSIS

Andrew Crews, Forrest Brewer

University of California Santa Barbara
crews@engineering.ucsb.edu
forrest@ece.ucsb.edu

ABSTRACT

In this paper, the problem of determining if a given sequential specification can be made to fit a predetermined set of shape constraints is explored. Shape constraints are constraints not on the logical makeup of the automata, but on the distribution of information required to support correct output behavior. These constraints allow the machine to be distributed into partitions with predefined communication and bit-level encoding complexity. For example, construction of a machine in the form of a feedback shift-register is a simple case of a shape constraint. Such constraints are based on the notion of a "D-atom" which is somewhat similar to dichotomy based encoding.

1. INTRODUCTION

The sequential encoding problem is one that has been researched for many years. Much of the work has involved techniques for logic minimization, which is often appropriate for isolated finite state machines (FSM's) without fixed locality of input and output. In this paper, we focus on the encoding issues pertinent to constructing distributed state machines. In this case, distributed issues such as wire delay and communication power dissipation or skew, dominate the design objectives. Such state machines occur as distributed controllers for VLSI designs in which floorplanning has been performed so that sensible metrics for communication cost and delays may be assessed. In such cases, communication costs between portions of the controller or between the controller and its inputs and outputs may dominate the design costs. For example, it is not uncommon for a critical timing path to pass through the control FSM. Design options including the potential for pipelining and/or locally regenerating critical signals can have significant benefits on the overall design. Typically, the input/output terminals of the controller are fixed temporally and spatially, while the encoding of the communication between blocks of the controller (state encoding) and distribution of the information flow in the machine may have considerable freedom.

In this paper, we identify an atomic piece of symbolic information within an FSM, called a *D-atom* (formally defined in section 2), and describe the dependencies that exist between such atoms, which are independent of the state encoding and depend only on the structure of the STG. Sets of D-atoms inherit dependencies from their element D-atoms. Such sets describe precisely what information is present or can be created within a block of a FSM corresponding to that set. Further, D-atom sets provide constraints on possible state encodings meeting these information requirements. Using D-atoms, we propose and implement a method of evaluating whether particular circuit topologies are possible for symbolic state machines specified as STG's. The topologies are constrained by the allowed communication signals and bitwidths between portions of the FSM's in terms of primary inputs, outputs and internal signals.

This paper is organized as follows: First, previous work is discussed as well as the relation of this work to that of dichotomy based encoding. In section 2, we describe the D-atom and related properties used throughout the paper. In section 3, we describe derivation of the symbolic dependency network for an FSM. Section 4 presents algorithms for evaluating properties of the state machines, independent of state encoding, such as construction within a given circuit topology, or given input/output placement restrictions. Section 5 describes constraints for encoding the precise symbolic dependency information desired. Finally, section 6 provides the results of this analysis for several benchmark circuits.

1.1 Previous work

It is impossible to provide a fair rendering of the previous work on state encoding given the brevity of this paper. Instead, we shall describe its relation to the most similar approaches, focusing on the differences. In this work, the minimal information needed to discriminate two different symbolic states is called a D-atom and we characterize the properties of sets of such elements. Closely related to this is the notion of a state dichotomy which originated with Tracey [11] and has seen recent renewed interest in Yang [13], Saldanha [9], Coudert [2] and others.

A dichotomy is an unordered pair of disjoint state sets. Any dichotomy can be encoded as a single bit encoding by choosing a 1 or 0 for each state of the two disjoint state sets respectively. In [2] Coudert showed a technique to find an encoding that satisfies many dichotomy constraints, including the *face* constraints [9], a necessary subset of the requirements for 2-level logic minimization. Dichotomies can be combined into larger dichotomies by making set unions of each pair as long as the result remains disjoint. In our formulation, although we merge multiple D-atoms into sets, such sets are not necessarily a dichotomy nor need they be decomposable into dichotomies. Since each D-atom may bring its own requirements into a set, we must represent each atom present, not the simpler disjoint dichotomy. This will allow a more general treatment of encoding and dependency analysis.

2. DEFINITIONS

First we define the D-atom, then clarify the state encoding method used within this paper. We also describe how to interpret the symbolic dependencies of a D-atom.

2.1 Shape Constraints

In the following, we are primarily concerned with construction of a distributed machine, where communication delays will determine the critical path, rather than logic complexity. A *shape constraint* is a constraint on the allowable communication (directional wires) between partitions of the machine. A shape

constraint is a fixed set of partitions of the machine, along with a description of the allowed communications between partitions. Note that the partitions are not forced to contain specific pieces of the machine, and their contents may range anywhere from the entire FSM, to empty.

In the worst case, each portion of the machine (for example, each bit) may be required to support every primary output and next state function. In fact, standard machine encoding techniques often produce encodings which exhibit low logic complexity, but worst case communication cost. We will use shape constraints and the techniques in this paper to determine whether an encoding exists for a specific set of allowable communications between partitions.

2.2 D-atoms and D-Sets

Definition: Given a set of state symbols, *S*, a *D-atom* is an unordered pair (s_i, s_j), $i \neq j$, $(s_i, s_j \in S)$, which implies s_i is distinguished from s_j.

Definition: In an encoding *E* over a set of states *S*, two states s_i and s_j are *distinguished* iff $E(s_i) \cap E(s_j) = \varnothing$.

An arbitrary set of D-atoms shall be called a *D-set*. In the standard method of encoding state machines, a unique minterm is assigned to each state. In that case, each bit represents a conventional dichotomy[2]. If we consider the encoding of a single bit b, for such an encoding, then b only distinguishes s_i from s_j if $E_b(s_i) = 1, E_b(s_j) = 0$, or vice versa.

In this paper, we are concerned with the encoding of an arbitrary set of D-atoms. In work done with dichotomies[2], only the question of whether these pairs (in our case, D-atoms) *could* be contained in a single dichotomy was considered (at the expense of adding additional pairs, and their dependencies to the set). Adding non-essential D-atoms to the set may incur additional dependency costs. Thus, we propose a more general method that encodes any D-set. Consider the example in figure 1:

Possible encoding for the set:

{ (S_1,S_3), (S_1,S_4), (S_2,S_4) }

State	*Code*
S1	01
S2	01,10
S3	00,10
S4	00

Figure 1. Example encoding for a set of D-atoms.

The three D-atoms would fit into a dichotomy, but only if a fourth, (S_2, S_3) were added, whose addition might require increased communication costs. In order to manage arbitrary D-sets, we allow encodings to consist of arbitrary functions, rather than restricting them to minterms. With this method, a D-set may be implemented in one or more bits in the final design while a D-set is a dichotomy only if it can be encoded in precisely one bit. Note that encoding a D-set in such a way that two states s_i and s_j are distinguished is equivalent to including the D-atom (s_i, s_j) into the set. So, for instance, in figure 1 since (s_1, s_2) is not in the set, those states must not be distinguished by the encoding, and this is achieved by having those states share the code "01".

2.2.1 Affects of State Encoding

We can view the representation of $s \in S$, now an arbitrary function of the state bits, as a modification of the STG, where a duplicate copy of state *s* exists for each minterm in the on-set of the function *fs*. Every input transition and every output transition are duplicated for every copy of state *s*. (See figure 2.) This representation is actually non-deterministic, since each transition to state *s* is now replicated several times, and the machine is free to decide which of the transitions to choose at any time. This freedom can be exploited when final encodings for the machine are decided.

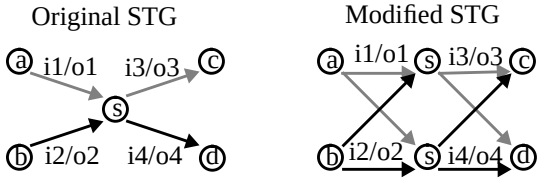

Figure 2. A modified STG. State s is duplicated.

Since these new states are functionally redundant, it is easy to see that the modified STG is equivalent to the original. When there is a choice of transitions to take, we know that no matter which transition is selected, the output and the next set of transitions are equivalent to those we would take in the original STG.

3. D-ATOM DEPENDENCIES

Any encoding containing a given D-atom has a certain set of dependencies associated with it. Once all the D-atoms have been encoded, these dependencies represent the possible logic support for the state functions (communication costs). For the determination of these dependencies, STG's with non-symbolic (encoded) inputs and outputs are considered, although it is easy to generalize the algorithms for symbolic inputs and outputs. (Essentially, D-atoms could be used to distinguish any two symbolic inputs or any two symbolic outputs in the same way that they distinguish two states.) Non-symbolic inputs and outputs are used here because they are common, and because algorithms dealing with them also apply to the problem of analyzing an FSM given a partial encoding of the states.

First, dependencies required by a single transition are examined. Next, this is expanded to find dependencies for a D-atom or primary output. Finally, the Symbolic Dependency Network (SDN) is formalized. Each transition in the FSM specification is a four-tuple, (I,PS,NS,O) which defines the next state and output of the machine for a given input and present state.

To realize each primary output of the FSM, we need to distinguish every transition for which the output is a logic '0' from every transition for which the output is a logic '1'. This creates a set of transition pairs for each output which must be distinguished.

Consider two transitions t_1 and t_2 taken from an STG:

t_1: i_1, s_1 -> s_1', o_1

t_2: i_2, s_2 -> s_2', o_2

If $i_1 \cap i_2 = \varnothing$, then there exists a set of primary inputs that can be used to distinguish t_1 from t_2. [7] presented an algorithm for finding the set of all possible support sets required to distinguish any two arbitrary logic functions. We denote the function representing the set of possible support sets for distinguishing i_1 and i_2 as $D(i_1, i_2)$.

The Boolean variable which distinguishes state s_1 from s_2 is simply the D-atom (s_1, s_2).

To distinguish t_1 and t_2, we need to be able to distinguish either i_1 from i_2 or s_1 from s_2. The required support to distinguish t_1 and t_2 is represented by the following function:

$$D(i_1,i_2) + (s_1,s_2) \qquad \text{(EQ 1)}$$

If s_1 and s_2 are the same, then we can only distinguish the transitions by the inputs. If the input functions i_1 and i_2 intersect, then we can only distinguish the transitions using the present state information. If the present states are the same and the input functions intersect, then we don't (and can't) distinguish t_1 and t_2, either because t_1 and t_2 are equivalent (if the output and next state are the same), or the transitions are non-deterministic. All dependencies will be in terms of a set or sets of primary inputs (which can distinguish inputs of the transitions) and D-atoms (which distinguish the present states of the transitions).

To determine dependencies of D-atom (s_1, s_2) it is necessary to distinguish every transition to state s_1 from every transition to state s_2. The required support for a D-atom is given by the Boolean function in EQ 2:

$$\prod_{t_i \in T_{NS}(s_1),\, t_j \in T_{NS}(s_2)} D(i_i, i_j) + (s_i, s_j) \qquad \text{(EQ 2)}$$

where $T_{NS}(s)$ is the set of transitions to state s, and i_i and s_i are the input cube and present state of transition t_i.

Furthermore, the dependencies for each output o, are found as the union of the dependencies required to distinguish transitions in the on-set of o, from transitions in the off-set, as shown in EQ 3:

$$\prod_{t_i \in T_{on}(o),\, t_j \in T_{off}(o)} D(i_i, i_j) + (s_i, s_j) \qquad \text{(EQ 3)}$$

where $T_{on}(o)$ is the set of transitions in the on-set of o, and $T_{off}(o)$ is the set of transitions in the off-set of o.

3.1 Symbolic Dependency Network

In a symbolic dependency network (SDN), each node represents either a D-atom, an input, or an output. Input nodes have no dependencies, output nodes have no dependents. Edges represent possible dependencies, where an edge from node n_1 to n_2 indicates D-set encoding of n_1 may require input from the D-set encoding of n_2. For each node, a set of cubes whose domain is the possible dependencies is used to represent all possible sets of dependencies. The set of dependency cubes is found for each node using EQ 2 and EQ 3 above.

Initially we find the dependency cubes for each of the outputs. During the next iteration, the dependency for any D-atoms used by those outputs is found. This cycle continues until no new D-atoms are introduced. In this way, unused D-atoms (which may occur in non-state minimal machines or incompletely specified machines) are never introduced or evaluated.

The resulting dependency network contains all possible support sets for *any* encoding, and can be therefore very valuable. If the dependency function indicates that D-atom (s_1, s_2) necessarily depends on (s_3, s_4), then any function which distinguishes states s_1 and s_2 must necessarily have a wire from a piece of the encoding which contains (s_3, s_4). Similarly, it is possible to determine which D-atoms require wires directly from particular primary inputs.

4. SHAPE-BASED ANALYSIS

The SDN provides an exact symbolic dependency map for an FSM--symbolic in the sense that the information is encoding-independent. This information is a powerful tool for analysis of the machine. In this section, we describe how to determine the existence of an encoding in the presence of shape constraints such as routing and input/output placement requirements.

The analysis, based on topology of the circuit implementation, will consist of applying three types of constraints: 1) Fixing the number of D-sets (sets of D-atoms) that are allowed. 2) Selecting the allowed directed interconnect between D-sets (including self communication) 3) Determining which primary inputs are allowed as input to each D-set.

The algorithm for applying the constraints proceeds as follows. Once a particular topology has been chosen, every D-atom and every primary output are placed into each D-set in the circuit. Initially, the only restrictions are the primary inputs. Next, for each D-set, we determine the entire set of primary inputs and D-atoms that are available for the support of D-atoms and outputs in the D-set given the shape constraints. This set is determined by the user defined constraints on the primary inputs, and by determining which D-atoms exist in D-sets that are allowed to send communication to the given D-set. The available support set is compared to the set of possible supports (taken from the SDN) for each node (D-atom and output) in the D-set. When the available support set does not intersect the any possible support of a node, that node is removed from the D-set (it cannot be maintained if the inputs to the D-set do not include its required dependencies). Removal of a node from one D-set may reduce the available support set for another D-set, so the process must be iterated until contents of the D-sets become stable.

Note that if all inputs are provided to each D-set, then the circuit will remain in its initial condition. This states simply that the entire machine may exist in a single D-set. In such cases where the shape constraints are very loose, we can apply additional constraints by removing D-atoms from particular D-sets and then iterating the algorithm above. The decision of which D-atoms to remove could be based on the required placement of outputs and the symbolic dependencies of those outputs. It could also be based on a pre-specified partial encoding of the machine provided by the designer. (Now we can view a partial encoding as simply a placement of D-atoms into D-sets.)

4.1 Analyzing the D-set Content

After iteration of the shape constraints, each D-set contains the maximal set of D-atoms and outputs that are allowed. The existence of an encoding is easy to determine from the D-set content. If each output exists in some D-set (or more specifically, exists in a D-set allowed for that output), then the machine is encodable. We will discuss the actual encoding in section 5.

Even when some outputs no longer appear in any of the D-sets, we may gain some valuable information about the machine: namely any output that *does* occur is realizable for the chosen topology. This information is useful in the redesign of the FSM, or in the determination of a new topology. Section 6 contains more information on the analysis of the D-set content.

5. D-SET ENCODING

Consider the problem of encoding a given D-set. Each D-set represents a partial encoding of the machine. Let $E_B(s)$ represent the partial encoding of state s in D-set B. For an arbitrary set B, we must ensure that:

$$(s_1, s_2) \in B \Leftrightarrow (E_B(s_1) \cap E_B(s_2) = \varnothing) \qquad \text{(EQ 4)}$$

EQ 4 implies two things which we will enumerate very carefully. 1. If $(s_1, s_2) \notin B$ then $E_B(s_1)$ *must* intersect $E_B(s_2)$ (which implies it is not distinguishable), and 2. If $(s_1, s_2) \in B$ then $E_B(s_1)$ must *not* intersect $E_B(s_2)$. If the first is violated, then (s_1, s_2) is not actually in the D-set, and so it is not provided as support to D-sets which receive communication from D-set B. If the second is violated, then we have included an additional D-atom in the D-set, which may require additional undesired communication. In the encoding of B, we must ensure the validity of EQ 4.

There is a trivial (but expensive) encoding which always guarantees EQ 4. Essentially, we could allocate one bit for every D-atom. This encoding results in a enormous amount of don't care information, since we may use as many as $n(n-1)/2$ bits to encode *n* states. With each bit containing exactly one D-atom, we have no trouble verifying that EQ 4 is true. (Note that this is the initial "seed" encoding specified in [4].)

It is also possible to determine an encoding which uses the minimal number of bits as follows: Create a graph with one node for every state, and an edge between every two states, s_1 and s_2 for which $(s_1, s_2) \in B$. An edge in the graph indicates that the encoding for the two states must not intersect, and the absence of an edge indicates that the encoding must share at least one minterm. It is by assigning sets of minterms to states according to these rules that we can encode D-sets in a small number of bits. At this point, the utility of encoding states as arbitrary functions becomes evident.

The following is offered without proof: if we consider the complement of the graph described above, then the minimum number of minterms can be assigned to the states in a legal encoding by assigning one minterm to each clique in the graph. The base two log of the number of minterms is the minimum number of bits required by the D-set. Any assignment of minterms to these cliques is a valid encoding.

This algorithm is given in figure 3.

Given:
- A set of states S
- A set of D-atoms, A

1. Create an empty graph G
2. Add 1 vertex to G for every state $s \in S$
3. Add 1 edge s_1-s_2 for each D-atom d = $\{s_1, s_2\} \notin A$
4. Obtain a minimal set of cliques C, which cover G
5. Assign a minterm to each clique
6. Encode each state as the union of the minterms correspondin the cliques which contain the state

Figure 3. D-set encoding algorithm

Note that the clique covering problem is, in general, NP complete, but solvable in polynomial type for k=2 (2 cliques, implying a single bit of encoding for the set) [3]. Additionally, there are many heuristic algorithms for clique covering with low time complexity.

6. RESULTS

In this section, a detailed analysis is done for three interesting machines. Following that is a table of more general analysis for a set of benchmarks from the 1993 Logic Synthesis Workshop.

The first example comes from the specification for machine "A" examined in [10] (see figure 4). In the specification the states are

PS	*I1*	*I2*	*I3*	*I4*	*out*
1	1	2	3	4	1
2	3	4	1	2	1
3	2	1	4	3	0
4	4	3	2	1	0

Figure 4. Hartmanis and Stearns' machine "A"

labeled 1, 2, 3, 4, and I1, I2, I3, and I4 denote symbolic inputs. In the paper, the authors note that this particular machine has a good two bit encoding which allows each bit to be a function of only the other bit and a single primary input. This example is labeled "A" in table 1. After applying these communication constraints to the FSM, the contents of each D-set is shown in figure 5. Two D-atoms (4,2) and (3,2) are allowed in both D-sets, but reapplying the constraints after removal of any single D-atom in either D-set results in removal of the output. Note that both D-sets can be represented by dichotomies, and hence bits. Thus, ignoring bit inversion, the analysis reveals that there is *exactly* one two-bit encoding of the machine under the above communication constraints.

D-set 1	D-set 2
(3,1), (3,2),	(2,1), (4,3),
(4,1), (4,2)	(3,2), (4,2), out

Figure 5. D-set contents for machine "A"

The next interesting example is "shiftreg" from the benchmark suite. Of all the benchmark files, shiftreg was the only file that fit entirely into a shift register construction. The shift register constraints assert that the only logic gates that will appear in the circuit will be inverters, and perhaps a single arbitrary function of the inputs at the first D-set. The "fit" of the FSM to the constraints is determined by the presence of all outputs in some D-set of the machine. Additionally, once the constraints were applied, it was evident that there exists only a single encoding, again ignoring inversions of the bits, that met the shape constraints.

This structure was further examined by removing some subsets of transitions from the graph, effectively replacing them with "don't care" transitions. This created a machine with only four states, but the three bit shift register implementation is still the smallest (in gates) of all logic implementation. Such an encoding is typically very hard to derive using standard logic minimizing encoding algorithms. Indeed for this four state machine, JEDI and NOVA state assignment tools could not find this smallest encoding even when forced to use a three bit encoding. However, by constraining the machine to the shape of a shift register, two things are obvious. First, the machine can be implemented as a shift register, and second, there exists exactly one encoding which fits the shift register shape constraint. The encoding is easy to derive from the D-set content, just as it was in the previous example.

The last example is count11, invented to exhibit some interesting properties. The FSM continually counts to eleven, and outputs a '1' on the eleventh cycle only if the input is a '1' during that cycle. Every node in the SDN depends only on other D-atoms, except for the output node which depends on both D-atoms and the primary input. The set of D-atoms that is recursively never dependent on primary inputs could be said to have purely sequential dependencies. The size of this set in each machine is labeled in the column "node clock" of table 1. The D-atoms which are purely sequential represent clocks that exist inside the machine, in some cases. However, in all the other FSM's non-zero values in the "node clock" column are actually the result of D-atoms that require no support because the STG specification is not strongly connected. Essentially they contain states which are unreachable except after resetting the machine to one of the weakly connected states. Note that these states are not redundant states, and the purely sequential D-atoms which appear are not the result of non-state minimal machine specifications. The D-atoms that represent distinguishing redundant states never appear in the SDN.

Table 1 contains some general analysis results for benchmarks FSM's, and three additional FSM's. The FSM "pipectl" at the bottom of table 1 is simple five stage pipeline controller. "PI" is number of primary inputs. "PO" is number of primary outputs. "S" is number of states. "node total" is the number of D-atoms and outputs. "node comb" is the number of D-atoms which can be

derived entirely from primary inputs (no other D-atoms required). "node clock" is the number of D-atoms that can be derived entirely from other D-atoms, and have no dependency on inputs, at any level. "max FSR" is the maximum depth linear feedback shift register (FSR) construction, in terms of D-sets, that could exist in the circuit and provide useful information. "max SR" is the maximum depth shift register, in terms of D-sets, that could exist in the circuit and provide useful information. "SDN sec" is the time required to construct the graph. "shape sec" is the time required to apply both the "max SR" and the "max FSR" constraints to the FSM.

TABLE 1. FSM Topological Characteristics

FSM	*PI*	*PO*	*S*	*node total*	*node comb.*	*node clock*	*max FSR*	*max SR*	*SDN sec*	*shape sec*
bbara	4	2	10	47	0	0	3	0	0.4	0.3
bbsse	7	7	16	124	21	39	3	2	0.4	0.7
bbtas	2	2	6	17	2	0	3	1	0.1	0.1
beecount	3	4	7	25	14	0	3	1	0.1	0.1
cse	7	7	16	127	33	0	3	1	1.3	3.8
dk15	3	5	7	26	10	0	2	1	0.7	0.0
dk16	2	3	27	154	143	0	3	2	6.7	30.4
dk17	2	3	8	31	10	0	3	2	0.3	0.3
dk27	1	2	7	23	8	0	4	2	0.1	0.2
dk512	1	3	15	108	32	14	5	3	0.4	2.6
ex1	9	19	20	209	16	0	5	1	6.1	10.6
ex4	6	9	14	100	0	13	7	0	0.4	0.5
ex6	5	8	8	36	3	7	2	1	0.5	0.3
keyb	7	2	19	173	16	0	5	4	2.1	9.1
mc	3	5	4	11	0	0	2	0	<0.1	<0.1
opus	5	6	10	51	9	0	2	1	0.4	0.5
s1	8	6	20	196	33	0	2	1	5.3	11.9
s1488	8	19	48	n/a	n/a	n/a	n/a	n/a	153	>1000
s208	11	2	18	155	0	18	16	0	3.2	7.7
s27	4	1	6	16	6	0	2	1	0.2	0.1
s386	7	7	13	85	31	0	3	2	1.2	1.9
s510	19	7	47	1088	0	0	19	0	15	1163
s820	18	19	25	319	15	0	4	1	26	35
sand	11	9	32	505	1	0	3	1	57	151
scf	27	56	121	n/a	n/a	n/a	n/a	n/a	>1000	NA
shiftreg	1	1	8	29	16	0	4	4	0.1	0.4
sse	7	7	16	70	21	31	3	2	0.8	1.8
styr	9	10	30	445	38	0	6	1	51	158
tav	4	4	4	10	0	6	1	0	0.4	0.0
tbk	6	3	32	n/a	n/a	n/a	n/a	n/a	>1000	NA
A	2	1	4	7	0	0	2	0	<0.1	<0.1
count11	1	1	11	56	0	55	1	0	<0.1	0.1
pipectl	1	5	47	1086	589	0	6	4	24	746

The FSR constraint allows inputs only in the first D-set, and allows each D-set communication only from the D-set immediately preceding it and any D-set following it. The SR constraint allows primary inputs to only of the first D-set, and each successive D-set is allowed only input from its immediate predecessor. The topology for these shape constraints is shown in figure 6. For an FSM, a maximum FSR depth of one indicates the topology is totally useless, while a maximum shift register depth of zero indicates that the shift register topology is useless. Other values imply that some portion of the circuit may fit into the constraints. As stated above, only one example (shiftreg) fit entirely into the SR constraint.

The FSR and SR models were used because they are highly restrictive shape constraints which are, in general, useful and interesting. Other such useful, highly restrictive constraints could have been created specifically for each benchmark, had we had more information about the benchmarks, such as the placement of its primary inputs and primary outputs.

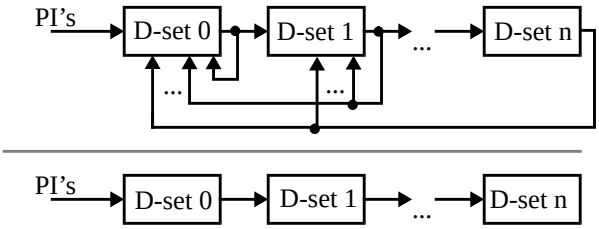

Figure 6. FSR shape constraints (top), SR (bottom)

7. CONCLUSIONS

For the small FSM cases, this technique enables a fine grain controllability that is missing from conventional tools, especially when timing and physical constraints need to be met. Further, shape constraints could be used to ensure that the controller synthesis meet timing and integrity constraints for sub-micron designs. However, the complexity of the described global analysis techniques definitely limit the general applicability of the current implementation. In future work, these limitations must be directly addressed and well as generalizations of shape constraints which could be used to limit the complexity growth.

8. REFERENCES

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," *27th DAC*, pp 40-45, June 1990.

[2] O. Coudert, C.-J. Shi, "Exact Dichotomy-based Constrained Encoding," *Proc. ICCD*, pp 426-431. Oct., 1996.

[3] M. Garey, D. Johnson, *Computers and Intractability*, New York, Freeman and Co., 1979.

[4] W. Grass, I. Lemberski. "Support Based State Encoding Targeting FSM Optimal LUT FPGA Implementation." *Proc. IWLAS*, pp 97-104, 1997.

[5] E. Goldberg, T. Villa, R. Brayton. "A Fast and Robust Exact Algorithm for Face Embedding," *ICCAD*, pp 296-303. Nov., 1997.

[6] Z. Kohavi, *Switching and Finite Automata Theory*, 2nd Ed. McGraw-Hill, New York, 1978.

[7] B. Lin, "Efficient Symbolic Support Manipulation," *Proc. ICCD*, pp 513-516. Oct., 1993.

[8] B. Lin, *Synthesis of VLSI Designs with Symbolic Techniques*, Ph.D. Thesis, UC Berkeley, UCB/ERL M91/105, Nov. 1991.

[9] A. Saldanha, T. Villa, R.K. Brayton, A Sangiovanni-Vincentelli, "Satisfaction of Input and Output Encoding Constraints," *IEEE Trans. CAD*, pp 589-602, May 1994.

[10] R. Stearns, J. Hartmanis, "On the State Assignment Problem for Sequential Machines II," *IRE Trans. on Elec. Comp.*, pp 593-603. Dec. 1961.

[11] J. Tracey, "Internal State Assignment for Asynchronous Sequential Machines," *IEEE Trans on Elec. Comp.*, pp 551-560. Aug. 1996.

[12] T. Villa, T. Kam, R. Brayton A Sangiovanii-Vincentelli, *Synthesis of Finite State Machines: Logic Optimization*. Boston, MA. Kluwer Academic, 1997.

[13] S.Yang M. Ciesielski, "Optimum and Suboptimum algorithms for input encoding and its relationship to logic minimization," *IEEE Trans. on CAD*, pp 4-12, Jan. 1991.