Simplification of Non-Deterministic Multi-Valued Networks

Alan Mishchenko
ECE Dept.
Portland State University, Portland, OR
alanmi@ee.pdx.edu

Robert Brayton
EECS Dept.
University of California, Berkeley, CA
brayton@eecs.berkeley.edu

Abstract[1]

We discuss the simplification of non-deterministic MV networks and their internal nodes using internal flexibilities. Given the network structure and its external specification, the flexibility at a node is derived as a non-deterministic MV relation. This flexibility is used to simplify the node representation and enhance the effect of Boolean resubstitution. We show that the flexibility derived is maximum. The proposed approach has been implemented and tested in MVSIS [16]. *Experimental results show that it performs well on a variety of MV and binary benchmarks.*

1 Introduction

The simplification of multi-valued (MV) logic networks is an important step in solving problems formulated in terms of MV logic. These include MV logic synthesis [9], software-hardware co-design [6][8], asynchronous circuit synthesis [10], and data mining [5].

Optimization of MV networks is achieved by applying a number of logic simplification and restructuring operations. One such method consists of deriving internal don't-cares for the logic functions of the nodes based on the particular network structure. These don't-cares are used to simplify the nodes' representations. A version of don't-care simplification for binary networks, developed in [17], uses a combination of satisfiability don't-cares (SDCs) and compatible observability don't-cares (CODCs). It was implemented as command *full_simplify* in SIS [1], with several improvements reported in [18]. Recently, it was shown that a CODC can be computed independently of a node's implementation [2]. The concept of CODCs was generalized for MV networks and successfully implemented [7] (*fullsimp* in MVSIS) and applied to MV network optimization. Like SIS, MVSIS can be applied to sequential circuits, FSMs and automata.

We provide a new scheme to compute the flexibility of a node in an MV network. The flexibility computed is complete (maximum) but not compatible. Hence, it must be used, to simplify the node, before any other nodes are changed. Because the flexibility is complete, it allows for a more thorough simplification compared to compatible or partial flexibility derived by other methods, which are subsets of the complete flexibility.

Although complete (or maximum) flexibility has been derived for Boolean networks where each node has a single binary output [19] and even for networks with multi-output binary nodes (Boolean relations) [20], no result existed for MV networks.

The present work differs from [7] in the following ways:

- the node representation before and after simplification can be non-deterministic;
- flexibility is derived as a non-deterministic relation;
- maximum (not compatible) flexibilities are used;
- "partial cares" are used for minimization;
- new methods for MV-SOP minimization, including a minimum non-deterministic cover, are developed.

These ideas have been implemented as a new MVSIS command, *complete_simplify*, which is more effective than other network simplification options in MVSIS [16], such as *fullsimp* [7].

The rest of the paper is organized as follows. Section 2 supplies necessary background in multi-valued logic. Section 3 presents an algorithm to compute the complete flexibility for a MV node in the MV network. Section 4 discusses the simplification of the nodes using this flexibility. Section 4.4 addresses fast computation of MV irredundant sums-of-product (MV-ISOP). Section 5 gives some experimental results. Section 6 concludes the paper.

2 Background

2.1 Multi-Valued Relations

Definition. Let the domain of a multi-valued variable a_i be denoted as D_{ai}. An *MV relation* R with MV input variables $\{a_i\}$ and an MV output variable z relates input minterms to output values:

$$R(a_1, a_2, \ldots, a_n, z): \; D_{a1} \times D_{a2} \times \ldots \times D_{an} \times D_z \rightarrow \{0,1\}.$$

Definition. The set of minterms of the combined input domains of relation R, where the output takes only one value, is the *care set* of R. The set of minterms, where the output is the set of all possible values, is the *don't-care set* of R. The set of other minterms is the *partial care set* of R.

Definition. If the total domain of R is the care set, R is a *completely specified* MV function. If the total domain of R consists of the care set and the don't-care set, R is an *incompletely specified* MV function. If the domain of R contains

[1] The first author was partially supported by a research grant from Intel Corporation. The second author acknowledges the generous support of the SRC under contract 683.004, the GSRC and the California Micro program with our industrial sponsors, Cadence and Synplicity.

as least one partial care minterm, R is a *partially specified* (*non-deterministic*) MV relation.[2]

Example. Figure 1 shows three ternary MV relations depending on binary variable a and ternary variable b. Relation R_1 is completely specified; R_2 is incompletely specified; R_3 is partially specified, or non-deterministic.

R_1

b\a	0	1
0	0	2
1	0	0
2	1	0

R_2

b\a	0	1
0	0,1,2	2
1	0	0
2	0,1,2	0

R_3

b\a	0	1
0	0,1	1,2
1	0,1	0
2	0,1,2	0,1

Figure 1. Example illustrating types of MV relations.

Definition. The i-th on-set (called *i-set*) of MV relation R is the MV-input binary-output function F_i defined over the input domain of R and taking value 1 for those minterms where the set of output values of R contains value i. The *essential i-set* consists of those minterms where the output value set consists of only value i.

Note that the i-sets of a completely specified MV function are pair-wise disjoint. The i-sets of an incompletely specified relation or partially specified relation can overlap. The essential i-sets are always disjoint.

Definition. MV relation R_1 *is contained in* (or *implies*) MV relation R_2 (denoted $R_1 \Rightarrow R_2$) if they have the same input domain and for each such minterm, the output values of R_1 are a subset of the output values of R_2.

Example. In Figure 1, $R_1 \Rightarrow R_3$ is true, but $R_2 \Rightarrow R_3$ does not hold because for the minterm (0,0), the value set of R_2, {0,1,2}, is not a subset of the value set of R_3, {0,1}.

Definition. A *multi-valued sum-of-products* (MV-SOP) is a representation of an MV function, in which each i-set of the MV function is represented by a set of MV cubes.

To reduce the total number of cubes needed to represent the MV-SOP, one i-set is selected as the default one and is not stored but is computed on demand by complementing the sum of the cubes belonging to the other i-sets.[3]

2.2 Multi-Valued Networks

Definition. An *MV network N* is a directed acyclic graph with nodes represented by MV relations.[4] The sources of the graph are the *primary inputs* of the network. There is one dummy sink whose inputs are the *primary outputs*.

Typically, the nodes and their output signals are named the same. The output of a node may be an input to other nodes called its *fanouts*. The inputs of a node are called its *fanins*.

Definition. An MV network is *non-deterministic* if any of its outputs is non-deterministic as a function of the primary inputs. If all internal nodes are deterministic, then the network is deterministic.

Actually, there are two notions of what constitutes a non-deterministic function of the primary inputs of a network. The first is what can be simulated when a ND node randomly selects one of a possible set of output values. Then, a network is ND if there exists a primary input minterm, which can cause different output values at one of the outputs. The second notion is similar, except at each internal node, the *set* of all possible values are propagated to its fanouts. In effect, on a fanout net, different values can propagate to different fanouts at the same time. This is similar to what is done in 3-valued simulation.

We use this second notion in this paper. The set of global relations obtained at the outputs is called the *behavior* of the network. The first notion is not used because it is not invariant under node elimination and collapsing. For example, if a non-deterministic node has several fanouts, and it is eliminated, the effect is as if several copies are made and then each is inserted into a fanout. After this there is no correlation between the values that these copies produce.

Definition. The Cartesian product of the MV domains of the fanin variables is the *local space* of a node. The MV relations of the nodes expressed in the local space are the *local relations*. The Cartesian product of the MV domains of the primary input variables is the *primary input space*. The MV relations of the nodes expressed in the primary input space are the *global relations*.

To compute the global relations, the network is traversed in a depth-first order and each local relation of a node is composed with the global relations of its fanins. The global relation of a primary input is the single-variable function of that input.[5]

Example. Consider the network given by the i-sets of its nodes:

```
x{0} = a{0}b{0}
x{1} = a{1}b{1} + a{0}b{1}
x{2} = a{1}b{1} + a{1}b{0}
m{0} = x{0}
m{1} = x{1}
m{2} = x{2}
n{0} = x{0}
n{1} = x{1}
n{2} = x{2}
{y}{0} = m{0}n{1,2} + m{0,1}n{2} + m{2}n{0,1} + m{1,2}n{0}
{y}{1} = m{0}n{0} + m{1}n{1} + m{2}n{2}
```

Node y is the PO; a and b are the PIs. Node x is non-deterministic. Two buffers, m and n, create two equivalent copies of the output of node x. The buffers feed into an equivalence detector y. Under the first notion of a non-deterministic network, the network is deterministic, since x=1 and x=2 have an identical effect on the output y if only one of them occurs at any time. However, collapsing the network in depth-first order gives:

```
{y}{0} = a{1}b{1}
{y}{1} = a{1}b{1}+ a{1}b{0} + a{0}b{1} + a{0}b{0}
```

which is non-deterministic. The reason is that during collapsing we substituted two independent copies of node x into node y, which allow for x=1 and x=2 to occur at the same time.

Definition. A transformation *changes the functionality* of the MV network if the global relation of at least one primary output is not contained in the original specification. Note that this definition does allow the set of behaviors represented by the network to decrease, as long as all the behaviors of the new network are contained in the original set of behaviors.

[2] In the binary domain, incompletely and partially specified are the same.

[3] In general, to be able to represent non-determinism at a node, all i-sets have to be represented.

[4] Normally, a network is represented by functions at the internal nodes.

[5] This procedure computes all behaviours of the network as defined by the second notion of non-deterministic networks.

In this paper, transformations are not allowed to change the functionality of the network. In general, it would be possible to consider transformations that do change the functionality if later they are followed by a transformation, which brings the network back into conformity, but this is not considered in this paper.

2.3 Flexibility at a Node

Definition. A *flexibility* at node y of an MV network N is an MV relation R^f_y such that replacing the current relation R_y of node y by any deterministic relation contained in R^f_y does not change the functionality of N. A flexibility R^f_y at a node y is *complete* if it is impossible to add another output value to any input minterm while preserving the flexibility property of R^f_y.

Definition. A set of flexibilities at a set of nodes is *compatible* if performing simultaneous replacement of the node relations by any set of relations contained in the respective flexibilities does not change the functionality of the network.

Any compatible flexibility at a node is a subset of the complete flexibility at that node. The flexibilities introduced and computed in [7] are compatible. They can be pre-computed and used independently at each node, but they are not complete and therefore may result in sub-optimal networks.

3 Flexibility Computations

The function at a node can be changed without adding to the behavior of the network by deriving the node's complete flexibility (CF) and replacing the current relation at the node by any *deterministic* function contained in this flexibility.

The computation of CF is done in two steps: first the CF is computed as a relation, $R(X,y)$, between the PI and the output of the node; second, it is computed as a relation between the inputs of the node, Y, and the output of the node, $R(Y,y)$.

Theorem 1. *Let N be a MV network and N^y be N clipped at node y. Let $R^i_j(X,y)$ be the global function of the i-set i of the primary output j, $1 \le j \le m$, of network N^y, expressed in terms of the PI X and the additional primary input y. Let $R^i_j(X)$ be the global function of i-set i at primary output j as derived from the initial specification of N, and let D_j be the domain of output j. The flexibility at node y in network N,*

$$R(X,y) = \prod_{j=1}^{m} \prod_{i \in D_j} [R^i_j(X,y) \Rightarrow R^i_j(X)]^6,$$

where the additional primary input y plays the role of the output variable of node y, is maximum.

Theorem 2. *Given the maximum flexibility, $R(X,y)$, at node y computed in the primary input space, and relation $M(X,Y)$ relating the primary input space to the local input space of y,[7] the flexibility in the local space,*

$$R(Y,y) = \forall_X [M(X,Y) \Rightarrow R(X,y)]$$

is maximum.

[6] This formula could be written in terms of relations instead of i-set functions using the correspondence

$$R(X,y,z_j) = \sum_{i \in D_j} [R^i_j(X,y) \Rightarrow (z_j = i)]$$

but the use of i-set functions seems simpler.

[7] It is important that $M(X,Y)$ be computed by eliminating internal nodes in topological order.

$R(X,y)$ is called the observability partial care (OPC). $\tilde{R}(Y,y)$ adds satisfiability don't cares to this. $M(X,Y)$ can be computed by an image computation using output co-factoring [7].

3.1 Resubstitution

Resubstitution is used to optimize an MV network by re-expressing a node's MV relation using additional inputs. The new representation of the node is accepted if the cost function of the node is reduced. As a result, the support size of F can change. Nodes that only fanned out to F but no longer do so, can be completely eliminated from the network.

Example. Let a, b, and c be ternary variables. Consider ternary functions F and G represented by MV-SOPs of their i-sets:

$$F^0 = a^{\{0,1\}}b^{\{2\}}c^{\{2\}} + a^{\{2\}}c^{\{2\}};\ F^1 = a^{\{0\}}b^{\{0\}};\ F^2 = \textit{default}$$
$$G^0 = a^{\{0,1\}}b^{\{2\}};\ G^1 = a^{\{2\}};\ G^2 = \textit{default}.$$

The representation of F can be modified without changing its functionality by incorporating G into the support of F:

$$F^0 = G^{\{0,1\}}c^{\{2\}};\ F^1 = a^{\{0\}}b^{\{0\}};\ F^2 = \textit{default}.$$

Even though the support of F has increased as a result of this resubstitution, the number of MV cubes and literals in the MV-SOP are both reduced.

To limit the amount of computation required for re-substitution at a node, we only consider those nodes, whose fanins are a subset of the fanins of the given node (subset support filter). In the above example, the support of G, is a subset of the support of F, and thus G is a candidate for re-substitution into F.

Resubstitution can be performed as an independent transformation without using flexibility due to the network structure. However, if a flexibility is available, the chances of finding improvement during re-substitution are enhanced. Therefore, in our implementation, similarly to [19], resubstitution is built into the node simplification process.

Theorem 3. *Let F and G be nodes with supports Y and B where B is contained in A. Let a flexibility of F in its local space be $R_F(Y,y)$, and the relation at node G in its local space be $R_G(B,g)$, where g is the MV variable of the output of G. The flexibility of F in the extended space, $Y_g = Y \cup g$, is:*

$$R_F(Y_g,y) = R_F(Y,y) + \overline{R_G(B,g)}$$

Theorem 3 is stated for one node, G, but for several nodes the flexibility is extended by summing the complements of the relations of these nodes.

After node simplification using the extended flexibility, the resulting MV relation may not depend on some of the input variables but may depend on some of the re-substituted variables. The choice to preserve or remove the dependence on a particular node is given to the node simplification procedure, which can make decisions motivated by the optimization criteria.

4 Node Simplification

We measure an MV-SOP by the total number of cubes in all of its i-sets. We present algorithms for computing small deterministic and non-deterministic MV-SOPs of an ND

relation and a method for computing the exact minimum non-deterministic MV-SOP of an ND relation.

4.1 Deterministic MV-SOP Minimization

The computation starts by ordering the i-sets heuristically. Typically, the current default value is ordered first. Then for each i, we extract the remaining minterms of its i-set not yet covered by i-set covers already computed. The minimized SOP for the i-set is computed by a call to an SOP minimizer using the remaining minterms that can't be covered by subsequent i-sets as the on-set and the rest of the remaining uncovered terms in the i-set as the don't-care set.

Since the remaining i-sets computed in each step do not overlap with the covers selected for the previous i-sets, the resulting MV-SOP is disjoint and, therefore, deterministic.

4.2 Heuristic ND MV-SOP Minimization

The computation proceeds in two steps. First, the essential part of each i-set is minimized using the rest of that i-set as don't-care. Computed this way, the i-sets are allowed to overlap resulting in a non-deterministic cover. This cover cannot be larger than the deterministic cover if we use the same ordering of the i-sets.

If at this point, all minterms are covered, the algorithm has computed the exact minimum cover (provided that the MV-input binary-output covers for each i-set have been minimized exactly). Surprisingly, in our experience, this is the case for about 90% of MV-SOP minimization problems that arise in the simplification of non-deterministic networks.

If there are remaining uncovered minterms, each must be associated with more than one output value. At this point, the algorithm determines if there is at least one output value common to all remaining minterms. In this case, all these are added to the common value, and if this i-set has the largest cover, it is made the default. This situation occurs in about 9% of the cases, leaving only about 1% to be processed further.

Finally, a simple greedy approach is taken. Considering values one by one in some heuristic order, as many minterms as possible are added to each of the successive i-sets.

4.3 Exact ND MV-SOP Minimization

Surprisingly, it is relatively easy to obtain an exact non-deterministic minimum cover. We first consider the case where there is no default value and the goal is to find a set of covers for all the i-sets which has the minimum total number of cubes. A minimum cover of a relation can be found as follows.

For each i-set, generate its set of primes. Form a global unate covering problem with the minterms to be covered being the entire input space and the union of all primes of all i-sets as the covering cubes. Solve for a minimum cover. Each prime chosen in the minimum cover is put into its appropriate i-set to form the minimum i-set covers.

Theorem 4: *The above procedure gives a set of i-set covers which has the minimum number of cubes. Each i-set cover is prime and irredundant.*

Note that the new relation is never larger than the current one. When the default i-set is used, it is never represented since it can be obtained by complementing all other i-sets.[8] The minimization problem is to choose the default in such a way that the remaining i-sets can be covered with the minimum number of cubes. This can be solved exactly as follows.

For each i, solve the covering problem as in Theorem 4 defined for all minterms of the input space that do not have value i in their value set. The measure of the solution obtained is the number of cubes in the cover, which does not contain the i-th set. Do this for each i and choose the solution that has the smallest measure.

Theorem 5: *The above procedure leads to the minimum set of covers when the default cover is not counted.*

4.4 Minimization Based on MV-ISOP

Runtime considerations in the first two algorithms (the exact algorithm has not been implemented yet) led us to experiment with other heuristic minimization options. A promising alternative was found in the Irredundant Sums-of-Product (ISOP) method of Minato-Morreale [14] using ZDDs [15].

This computation is applied to a binary-encoded MDD when an i-set is minimized with its don't-care. The result of the ISOP computation is the ZDD representation of the binary cover, which can be decoded back into MV cubes. This algorithm is fast but may not result in a prime or irredundant cover. For example, if one or more input variables have an odd number of values, then the binary ISOP is not prime and irredundant in the MV domain.

Binary ISOP can be used also as a preprocessing step to reduce the cube covers used in initial calls to Espresso-MV. Fast binary ISOP computation proved to be helpful when the initial specification had many cubes.

5 Experimental Results

The flexibility computation and simplification algorithms (except for the exact algorithm of Section 4.3) have been implemented in the MVSIS environment as the command *complete_simplify*. The comparison of *complete_simplify* with *fullsimp* [7] on the MV benchmarks was done on a 933MHz Pentium III PC under MS Windows 2000. MVSIS used less than 30Mb of RAM.

Benchmark	in	out	lat	node	ival	oval	BDD
4-arbit-cell.mv	4	3	5	190	2.4	2.3	158
bakery-proc.mv	9	3	3	258	2.8	3.5	219
coherence-cch.mv	10	4	6	240	2.4	2.8	399
coh-dir.mv	6	8	11	653	2.5	2.2	327
comp.mv	4	2	0	11	2.0	2.0	53
ele-ctr-det.mv	14	4	12	1446	2.0	2.0	276
eisenberg-proc.mv	5	2	2	284	4.3	5.8	206
matmul.mv	8	4	0	4	3.0	3.0	117
slider-nsf.mv	9	9	0	316	9.0	9.0	526
sort.mv	8	8	0	24	3.0	3.0	304

Table 1. Benchmark statistics.

[8] In binary logic synthesis, we usually only implement the onset of a node; if the offset is required, it is produced by an invertor.

Table 1 lists statistics of the benchmarks tested. The columns "in", "out", "lat", and "node" list the number of primary inputs, primary outputs, latches, and nodes in the network. Columns "ival" and "oval" give the average number of values in the domains of the primary input and output variables (including also the latch inputs and outputs). Column "BDD" shows the number of BDD nodes in the shared BEMDD computed for all nodes in the network after reading in the benchmark, building all global functions, and performing BDD variable reordering.

Table 2 illustrates the performance of *complete_simplify* on the benchmarks. Column "orig" lists the number of literals in the SOP after sweeping. Columns "fs", "mfsi", and "mfs" give the number of literals in the SOP after running, respectively, *fullsimp* [7], *complete_simplify* using ISOP, and *complete_simplify* using Espresso-MV as the MV-SOP minimizers. The runtimes compare *fullsimp* and *complete_simplify*, when both programs use Espresso-MV. The non-deterministic option in *complete_simplify* is used. In light of these observations, the runtime is surprisingly small. Although Espresso always gives better results than ISOP when minimizing a node, the overall minimization with ISOP is more efficient because ESPRESSO times out more often.

Benchmark	Literal count in SOP				Runtime, s	
	orig	fs	mfsi	mfs	fs	mfs
4-arbit-cell.mv	136	82	86	84	0.2	0.4
bakery-proc.mv	540	256	231	234	0.9	1.5
coherence-cch.mv	755	356	302	291	0.9	1.9
coh-dir.mv	774	368	335	371	0.8	1.6
comp.mv	289	18	8	8	0.2	0.1
ele-ctr-det.mv	664	231	243	238	0.9	1.2
eisenberg-proc.mv	746	462	286	322	7.7	6.0
matmut.mv	480	320	320	320	0.1	0.1
slider-nsf.mv	312	312	306	306	0.3	1.0
sort.mv	2296	174	174	174	6.6	1.3
Total	**6992**	**2579**	**2291**	**2348**	**18.6**	**15.1**
Ratio, %	**100**	**36.9**	**32.8**	**33.6**		

Table 2. Literal count and runtime comparison.

Benchmark	Literal count in SOP				Runtime, s		
	orig	ss	ssmi	ssm	ss	ssmi	ssm
4-arbit-cell.mv	136	54	55	57	0.9	1.5	1.8
bakery-proc.mv	540	160	154	157	38.2	4.6	10.3
coherence-cch.mv	755	152	141	139	2.4	2.9	8.3
coh-dir.mv	774	211	176	171	5.6	7.6	9.1
comp.mv	289	8	6	6	2.1	0.6	0.6
ele-ctr-det.mv	664	188	179	181	10.1	10.0	10.4
eisenberg-proc.mv	746	126	141	141	8.0	1.4	8.3
matmut.mv	480	96	112	112	0.4	0.7	0.7
slider-nsf.mv	312	216	222	222	5.1	1.6	2.9
sort.mv	2296	-	202	202	-	7.2	11.8
Total	**6992**	**1211**	**1186**	**1186**	**72.8**	**30.9**	**52.4**
Ratio, %	**100**	**17.3**	**16.9**	**16.9**			

Table 3. Comparison as part of optimization script.

Table 3 compares the performance (in terms of runtime and literal count) of *fullsimp*, *complete_simplify* with the ISOP option, and *complete_simplify* with the Espresso-MV option when these are used as part of an MVSIS optimization script *script.mvsis* [16] similar to *script.rugged* used with SIS. Column "orig" shows the original number of SOP literals after sweeping. Columns "ss", "ssmi", and "ssm" show the number of literals after running the script with *fullsimp*, *complete_simplify* using ISOP, and *complete_simplify* using Espresso-MV as MV-SOP minimization procedures. The dash in Table 3 indicates the script did not complete in five minutes. The average parameters have been computed without considering the last line. All final results were formally verified by comparing against the original circuits.

Table 2 shows that using complete flexibilities compared to CODCs (*complete_simplify* vs. *fullsimp*) enhances the quality of the final result. However, Table 3 shows this advantage is diminished when run inside a standard script. On the other hand, runtimes are improved by using complete flexibilities, and this seems to lead to a more rugged script. The runtime reduction is possibly explained by *complete_simplify* achieving a significant reduction early, allowing later procedures to run faster.

Table 4 shows the relative amount of flexibility due to SDC, CODC and CF measured as follows:

$$P = 100\% \times \frac{T - M}{M(V-1)},$$

where T is the sum total of the numbers of output values for all the input minterms of the relation, M is the number of the input minterms, and V is the number of values in the output range. The amount of flexibility is equal to 0% for completely specified functions and 100% for relations that can take all values for any minterm.

To illustrate how the amount of flexibility is measured consider Figure 1, where $M = 6$ and $V = 3$. The ternary function on the left has $T = 6$, which yields $P = 0\%$. The relation in the center and on the right have $T = 10$ and $T = 12$, which yields the amount of flexibility $P = 33\%$ and $P = 50\%$, respectively.

Benchmark	Global space, %		Local space, %		
	CODC	CF	SDC	SDC +CODC	CF
4-arbit-cell.mv	7	38	7	7	15
bakery-proc.mv	12	68	17	20	36
coherence-cch.mv	6	64	17	17	26
coh-dir.mv	2	61	18	18	31
comp.mv	0	3	17	17	19
ele-ctr-det.mv	12	42	7	7	15
eisenberg-proc.mv	1	57	38	38	53
matmul.mv	0	0	0	0	0
slider-nsf.mv	0	2	0	0	0
sort.mv	0	0	49	49	49
Total	**40**	**335**	**170**	**173**	**244**
Ratio, %	**12**	**100**	**70**	**71**	**100**

Table 4. Comparison of the amount of flexibility.

The numbers in Table 4 are averaged over all nodes in the MV network. Note that SDCs can only be measured in the local space. All initial circuits are deterministic, so the initial flexibilities are zero. Table 4 shows CODCs give little additional flexibility in the local space compared to SDCs, while the complete flexibility (CF) contributes 29% on top of SDCs + CODCs. It is not clear why CODCs adds so little compared to SDCs. The contribution of CODCs is different for binary benchmarks [11].

Table 5 compares *full_simplify* in SIS (columns "FS") with *complete_simplify* re-implemented in the SIS environment (columns "CS") using MCNC benchmarks. Column "Benchmark" gives the benchmark name. Column "Statistics" gives the number of inputs and outputs. Performance is measured using the sum of literals in all factored forms as reported by SIS. Column "Orig" gives the literals in the original benchmarks. Runtime in seconds is reported in the last section. Since SIS timed out for "alu4.blif", this was not included in the rows "Total" and "Ratio" of the table.

Table 5 shows that *complete_simplify* yields 11% less literals compared to *full_simplify* in SIS, while the runtime of the SIS-based re-implementation of *complete_simplify* is comparable to that of *full_simplify* in SIS. We are currently working on specialized techniques to speed-up the most time-consuming part of *complete_simplify*: repeated computation of the global BDDs. Preliminary experiments show that the new implementation will be approximately three times faster than the one reported in Table 5. This will make it applicable to circuits beyond the scope of SIS.

Benchmark	Statistics		Literals in FF			Runtime	
	Ins	Outs	Orig	FS	CS	FS	CS
9symml.blif	9	1	277	270	190	1.2	1.8
alu2.blif	10	6	453	374	415	1.9	2.4
alu4.blif	14	8	855	t/o	796	t/o	6.4
dalu.blif	75	16	3067	2331	1701	145.9	4.5
des.blif	256	245	6101	5677	4676	16.2	110.9
frg2.blif	143	139	2010	1522	1396	9.5	11.0
pair.blif	173	137	2420	2203	2131	6.5	25.9
c1908.blif	33	25	1497	1406	761	68.1	96.7
c432.blif	36	7	372	335	288	4.6	3.7
c880.blif	60	26	703	687	624	2.3	3.6
Total			**16900**	**14805**	**12978**	**256**	**266.9**
Ratio, %			**100**	**88**	**77**	**100**	**104**

Table 5. Comparison of *full_simplify* and *complete_simplify*.

We have not given the results where *complete_simplify* forces each MV-SOP to be deterministic since this option leads to significantly inferior quality. This is not a problem in binary networks; all relations are deterministic since at each node, only one *i*-set is represented, while the other is implied as the complement of the first.

6 Conclusions

The main distinctive aspect of this work is the use of full flexibility in minimizing nodes in an MV network. Experimental results demonstrate that the new approach is practical and produces compact representations when compared with other methods. Used as part of a standard MV logic optimization script, the new *complete_simplify* procedure leads to faster and more rugged processing. In general, our experience is that the use of MV networks helps reveal alternatives and generalizations, allowing for better understanding, even in the case of binary networks [12].

These methods are based on computing global BDDs. Beyond that, a possibility is to use partitioning so that only a part of the circuit is optimized at any one time [4]. Future work will try partitioning in the context of MV networks.

References

[1] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Dordrecht, 1984.

[2] R. K. Brayton. Compatible Observability Don't-Cares Revisited. *Proc. of IWLS'01*. pp. 121-126.

[3] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Comp*, Vol. C-35, No. 8 (August, 1986), pp. 677-691.

[4] J. Cong, Y. Lin, W. Long. SPFD-Based Global Rewiring. Prog. FPGA, 2002.

[5] C. M. Files, M. A Perkowski. Multi-Valued Functional Decomposition as a Machine Learning Method. *Proc. of ISMVL '98*, pp. 173 -178.

[6] J.-H. Jiang, Y. Jiang, and R. Brayton. An Implicit Method for Multi-Valued Network Encoding. *Proc. of IWLS'01*, pp. 127-131.

[7] Y. Jiang and R. Brayton. Don't-Cares and Multi-Valued Logic Network Optimization. *Proc. ICCAD'00*. pp. 520-525.

[8] Y. Jiang and R. Brayton. Logic Optimization and Code Generation for Embedded Control Applications. *Proc. of CODES' 01*, pp. 225-229.

[9] T. Hanyu, M. Kameyama. A 200 MHz Pipelined Multiplier Using 1.5 V-Supply Multiple-Valued MOS Current-Mode Circuits with Dual-Rail Source-Coupled Logic. *IEEE J.Solid State Circuits*, vol. 30, no 11, pp. 1239-1245, Nov. 1995.

[10] A. Kondratyev. Design of Delay-Insensitive Combinational Logic through MV-Synthesis. *Unpublished manuscript.*

[11] A. Mishchenko. An Experimental Evaluation of Algorithms for Computation of Internal Don't-Cares in Boolean Networks. Technical Report. Sept. 2001. http://www.ee.pdx.edu/~alanmi/research/net/DCcomparison.pdf

[12] A. Mishchenko, R. K. Brayton. A Boolean Paradigm in Multi-Valued Logic Synthesis. *Proc. IWLS '02*, June 2002.

[13] A. A. Malik, R. Brayton, A. R. Newton and A. Sangiovanni-Vincentelli. Reduced Offsets for Two-Level Multi-Valued Logic Minimization. *Proc. of DAC' 90*, pp. 290-296.

[14] S. Minato. Fast Generation of Irredundant Sum-of-Products Forms from Binary Decision Diagrams. *Proc. of SASIMI'92 (Synthesis and Simulation Meeting and International Interchange)*, Kobe, Japan, pp. 64-73.

[15] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. *Proc. of DAC '93*, pp. 272-277.

[16] MVSIS Group. *MVSIS*. UC Berkeley. http://www-cad.eecs.berkeley.edu/mvsis/

[17] H. Savoj. R. K. Brayton. The Use of Observability and External Don't-Cares for the Simplification of Multi-Level Networks. *Proc. of DAC' 90*. pp. 297-301.

[18] H. Savoj. Improvements in Technology Independent Optimization of Logic Circuits. *Proc. of IWLS'97*.

[19] H. Savoj. Don't Cares in Multi-Level Network Optimization. Ph.D. dissertation, UC Berkeley, May 1992.

[20] E. Sentovich, et al. "SIS: A System for Sequential Circuit Synthesis", *Tech. Rep. UCB/ERI, M92/41*, ERL, Dept. of EECS, Univ. of California, Berkeley, 1992.

[21] Y. Watanabe, L. Guerra and R. K. Brayton. Logic Optimization with Multi-Output Gates. *Proc. ICCD '93*, pp. 416-420.