Optimization of Multi-Valued Multi-Level Networks

M. Gao, J-H. Jiang, Y. Jiang, Y. Li, A. Mishchenko*, S. Sinha, T. Villa**, and R. Brayton
Electrical Engineering and Computer Sciences Dept.
University of California, Berkeley CA 94720
* Portland State University, Portland OR
** Parades, Rome Italy
(mvsis-devel@ic.eecs.berkeley.edu)

Abstract

A program called `MVSIS` *has been developed which optimizes multi-level multi-valued networks (MV networks). We describe what such a network is and the capabilities contained in* `MVSIS`. `MVSIS` *is modeled after* `SIS`*, which synthesizes binary multi-level networks, but the logic network of* `MVSIS` *is such that all variables can be multi-valued each with its own range. Included in* `MVSIS` *are almost all the technology-independent transformations of* `SIS` *for combinational and sequential logic synthesis as well as transformations specific to multi-valued nodes such as* `merge, pair_decode, encode`. `MVSIS` *can read and write* `BLIF-MV` *and* `BLIF` *files which describe MV-networks and binary networks respectively.*

1 Introduction

Multi-level multi-valued (MV) logic synthesis can have many applications including:

1. Logic synthesis for multi-valued hardware devices such as current-mode circuits [7].

2. Initial manipulation of a hardware description before it is encoded into binary and processed by standard binary logic synthesis programs [8]; MV is a natural way to describe procedures at a higher level.

3. A front end to a software compiler, since software lends itself naturally to the evaluation of multi-valued variables in a single cycle [10]. Strong logic synthesis transformations can be applied to compilers aimed at embedded applications.

4. Asynchronous synthesis of delay insensitive (DI) circuits [11].

5. Data mining, where the objective is a simple description that summarizes the content of some data [4].

We have developed and included techniques for optimization of MV networks. `MVSIS` [2] is an interactive tool, and has been made similar to `SIS` [19]. In the sequel, the components of `MVSIS` that are specific to a multi-valued environment and are different from their binary counterparts are described. Although details of the MV operations are beyond the scope of this paper, we include several examples illustrating the use of `MVSIS` for optimizing networks in order to present its main capabilities and to give ideas for its uses.

2 Network Specification

2.1 MV-Networks

An MV-network is a network of nodes; each node represents an MV-function with a single multi-valued output. The functions associated with each value (value-functions) of a node are stored in SOP form. We call these *i-sets*, e.g. the 0-set is the onset of the function where the node has value 0. There is one MV variable associated with the output of each node. A directed edge connects from node i to node j if any of the i-sets at node j depends explicitly on the variable associated with node i. The network has a set of primary inputs (all of which may be multi-valued) and a set of nodes, designated as the outputs of the network. An important distinction with other MV methods, is that each variable can have its own range, which can in particular contain two values. For each node, one of its i-sets is designated as the default value and is not stored. It can be recovered by complementing the sum of all the other i-sets.

In the initial specification, we allow non-deterministic relations at the nodes. This is done by allowing a minterm to be part of several i-sets. This may result in one or more of the primary outputs to be non-deterministic as functions of the primary inputs. In this case, the result of synthesis may be a subset of the initial relation specified.

MVSIS supports sequential MV-networks with multi-valued latches, i.e. storage devices that can hold any of a set of values.

2.2 External Don't Cares

External don't cares can be specified as a separate combinational network, called EXDC. The inputs of EXDC are the primary inputs plus the latch outputs; the outputs of the EXDC can be a subset of the primary outputs. The outputs of the EXDC are matched by name with those of the original network. They are binary variables, even though the original outputs may be multi-valued. Given an output function f, its companion external don't care function defines a Boolean function, such that for any of its onset minterms, the output of f can have **any** value (universal don't care minterms).

For any primary output, if all its i-sets are specified and their sum does not cover all the minterms in the input space, it is incompletely specified. In this case, the unspecified minterms are assumed to be able to take any value at the output, namely they are don't cares. In some applications, like data mining, the unspecified minterms can consume a large space. MVSIS provides a way of automatically extracting incomplete specification as external don't cares. In the current implementation, if an incompletely specified table in the network is (i) a primary output, and (ii) all its fanins are primary inputs, then an EXDC network is extracted for that node.

External don't cares are a form of non-determinism, but are restricted to the case when either an input minterm assumes a unique output value or any output value is allowed (universe of the output variable). MVSIS supports specification of partial cares, where an input minterm may assume any value from a proper subset of the universe of the output values. This is done by specifying a node whose i-set functions intersect. Also, we derive partial cares from the network structure and use them at internal nodes as part of a new minimization procedure.

External sequential don't cares can be derived from the network by extracting states that can't be reached from the specified initial state. These are combined with any already existing don't care network.

3 Combinational Optimization

3.1 Node Simplification

The i-sets (one for each output value) at an MV-node can be simplified using various simplify commands. Generally these use a two-level logic minimizer like ESPRESSO-MV [17], which minimizes MV-input, binary-output functions. The objective of a general two-level minimization is to find a logic representation with a minimum number of implicants (cubes) and literals while preserving functionality. Don't cares derived from the surrounding network structure can be used in the minimization process. Each of the i-sets, except the default, is simplified and replaced with simplified versions if the new functions have been improved according to the cost function in use. Recently, we implemented a multi-valued version of ISOP minimization [12] and found that it can be particularly effective when minimizing functions with large don't care sets. It is also helpful as a preprocessing step to ESPRESSO-MV.

For each node, an i-set is selected as the default. For example, for a binary output function, the offset is usually the default. The default i-set is never looked at unless a particular command requires it. For example, if the output x of a binary function is used in the complemented form $\overline{x}$ in a fanout, and the node producing x is eliminated, then the SOP for $\overline{x}$ must be computed and substituted in that fanout.

A powerful node simplification called `fullsimp` and is the direct generalization of the one in SIS. The notion of compatible observability don't cares (CODC) used in SIS [18] has been generalized to take MV-nodes into account [9]. Given these, MV-image computation techniques are used to map them to the local space of each node. In addition, a SDC of those nodes in the network whose support is a subset of the support of the node being simplified is added to the local don't care set thus derived. This allows a form of Boolean substitution when `fullsimp` is executed. Each node is then simplified by ESPRESSO-MV using this local don't care set.

A more powerful node simplification method [14], called `fullsimp_complete` performs the same steps as `fullsimp` (deriving flexibility and simplifying the nodes) but does it with the following differences:

1. The flexibility at a node is represented as a relation[1] between the node's fanins and its output (generally multi-valued). This relation gives all possible combinations of inputs and outputs of the node, which, when they appear at the node, will not change the overall network behavior at the primary outputs. It is a *complete* description of a node's flexibility.

2. The flexibility computation and node simplification are interleaved. The reason for this is that the complete flexibility is not compatible; thus a node must be optimized immediately after the flexibility is computed.

[1] In the multi-valued output case, this relation can describe "partial cares" which state that for a given minterm, the node output can be any of a subset of values. Note that for the binary output case, a partial care is the same as a don't care since any subset of values with more than one value is the full set, and hence a don't care.

When a node is modified, the changes are introduced into the network before the complete flexibility of the next node is computed.

3. Node representations before and after simplification are allowed to be non-deterministic. Having a non-deterministic node before simplification is not a problem because the flexibility relation computed at a node always contains the node representation, which can also be a relation. Allowing for a non-deterministic representation after simplification can reduce the literal count in the node representation.

4. The default value may be changed if this improves the cost function of the network. In the binary case, changing the default corresponds to a phase assignment step at the node, which is not performed in SIS.

5. New heuristic MV-SOP minimization methods, which allow for non-determinism of the resulting representations, have been developed for use with this new procedure.

3.2 Algebraic MV Methods

An important step in network optimization uses algebraic methods for extracting new nodes representing logic functions that are common factors of other nodes. Several techniques based on algebraic decomposition are part of `SIS`. Similarly, we have developed new algebraic techniques for MV-logic [1, 5, 6] which treat binary and multi-valued variables uniformly. These include methods for finding common sub-expressions, semi-algebraic division, decomposing a multi-valued network, and factoring a SOP form. Brief descriptions of these are listed below.

1. **Algebraic substitution** of one node into another is performed in `MVSIS` using semi-algebraic division [6]. This method attempts to divide or substitute one function into another. All i-sets of the divisor are tried. There are two modes for this. If the divisor is a two-cube divisor, then a fast method based on matching is used; otherwise, a slower branch and bound method (called the satisfiability-matrix method) is used.

2. **Node extraction** looks at all the nodes in the network and tries to extract good common factors and create new nodes in the network, re-expressing other nodes in terms of these newly introduced nodes. It is one of the transforms used to break down large functions into smaller pieces.

   Generally, the method works by generating candidate two-cube divisors by making each pair of cubes in a node's SOP i-set, cube-free. These candidate two-cube divisors are made canonical and hashed into a table. A count is kept on the number of hits for each entry, to obtain the value of a divisor. Complements of candidates are also kept if they are two-cube expressions. The divisor with the largest value is extracted as a new node and substituted (divided) into all functions where applicable. For efficiency, the candidate divisor table is only incrementally updated after each substitution.

3. **Factorization** creates a factored form for each i-set. The following is an example of a resulting factored form.

```
a{2,3}b{0,1} + a{0,3}b{1,2}
+ a{1,2}b{0,3} + a{0,1}b{2,3} =
(a{0,2,3}b{0,1,2} + a{0,1,2}b{0,2,3})
(a{1,2,3}b{0,1,3} + a{0,1,3}b{1,2,3})
```

   Factored forms of i-sets are used in evaluating the complexity of an i-set in some network cost functions.

   Decomposition uses factoring of each i-set of a node, but instead of creating a factored form for each, creates new nodes according to its factorization. Such intermediate nodes may not have been produced by node extraction, so there is a possibility of finding better common factors as well. After decomposition, a resubstitution followed by an elimination of useless nodes is done to eliminate duplicate factors.

3.3 Network Manipulations

1. **Collapsing** converts the entire multi-level network so that the SOP forms for each output are in terms of the primary inputs only. Thus the number of nodes in the network will be exactly the number of primary outputs. A new version of collapse is based on building the MDDs of the outputs, and deriving an ISOP [12] for each value. Generally, this is very fast if the MDDs can be built efficiently. In addition the use of ISOPs gives a result that is partially minimized.

2. **Merging** is an operation unique to the multi-valued domain. It takes a list of nodes and forces a *merge* of them into a single multi-valued node by building one i-set for each combination of values of the nodes being merged. In the worst case, if for example, there are k binary nodes in the list, it will create a single node with 2^k values. However, some new i-sets may be empty, in which case they are not created. In addition, if a pair of values always appears together in all the fanouts, then their functions will be combined into a single i-set. Merging can be made automatic by asking `MVSIS` to find good combinations of nodes to merge. Merging is one of the methods for creating MV intermediate nodes. Note that node extraction and decomposition

discussed in the previous sub-section only create binary output nodes, since these methods are based on AND/OR factoring.

3. **Encode** is like the inverse of the merge of binary functions. It tries to find a good binary encoding for each multivalued variable in the network, including primary inputs and outputs. At the end, each signal is encoded as a binary signal. Then a binary file can be written. However, often we want to keep the I/Os the same (e.g. for verification purposes), so as an option, encoders and decoders can be put at the inputs and outputs which keep the network in its original multi-valued I/O form.

   The encoding has two phases. The first starts at the inputs and for each node, determines if one of its fanins can be used to partially encode the node [15]. To achieve this, we test how many values are in each of the cofactors with respect to the chosen fanin (which can be multi-valued). If the number of values is reduced enough, the node is broken down into two functions, the chosen fanin (a node which already exists) and a new function that has fewer values. The effect after the first phase is a *partially encoded* network which has the same number of nodes, but fewer values at some of the nodes. The second phase starts from the outputs and in reverse topological order works back to the primary inputs. At each node, its outputs are encoded using the information on how its fanouts are used [8].

4. **Pair decoding** is a type of node extraction operation which does "bit" pairing to create a new multi-valued node. It is similar to merging, except for the criteria for choosing when to pair (or merge) two nodes. It looks for the "best" pair of signals to merge together, based on how these signals are used in the fanouts. Then like merging it creates a new node with values equal to all the decodes of the pair. The new node created is algebraically substituted into other nodes. Finally, any set of values of the new node, which always appear together in the fanouts, are merged into a single value of the new node. After this step, a simplify operation should be executed to effect the full substitution.

5. **Bi-decomposition** is another operation in `MVSIS` that creates multi-valued intermediate nodes [16]. It takes a flattened or partially flattened MV-network and generates another one composed of two-input multi-valued MAX and MIN gates and multi-valued literals. Both the incompleteness of the initial specification and the flexibilities generated in the decomposition process are exploited. Bi-decompositoin can be viewed as a mapping technique resulting in a network of multi-valued primitives analogous to NAND and NOR gate networks used in binary synthesis. This method is particularly suited for data mining because the maximum and minimum relations are easily understood by the humans.

4 Sequential Optimization

Similar to SIS, `MVSIS` performs all combinational optimizations on the combinational part of the sequential network. For a network with latches, `MVSIS` can compute the set of states which can be reached from an initial state. The set of unreachable states is then stored in the external don't care network (EXDC) and used in node minimization. In the case where there exists an EXDC network already, the unreachable state don't cares are ORed with the existing ones.

5 Other Operations

The usual operations which read and write `BLIF-MV` files, and print statistics about the network are part of `MVSIS`. One that is special to the multi-valued environment is `print_range` which lists all the nodes by name and the number of values for each signal.

6 Verification

MV-networks can be verified in `MVSIS` by either simulation or by formal methods. Validation refers to checking the equivalence of two networks by simulation. Formal verification computes the global function for each output using an MDD representation and compares the MDD structure; for sequential networks, it performs the same computation for each latch input as well. If a match cannot be found among the latch variables of the two networks to be verified, no verification is claimed by this method. `MVSIS` supports optimization of non-deterministic networks [14]. In these cases formal verification checks for containment instead of equivalence.

Sometimes it is important to know if a network is non-deterministic. `MVSIS` has a built-in incomplete test for non-determinism at the primary outputs which uses random simulation. If a network is non-deterministic and this non-determinism is detected by one of the random vectors, the network is declared non-deterministic; however, absence of a message does not imply that the network is deterministic.

7 Examples

A good way to understand `MVSIS` and what it might be used for is through examples. We include examples that illustrate various operations previously discussed.

7.1 Example 1

The specification of the example `matmul` is given below in the `BLIF-MV` format (see `BLIF-MV` documentation in `VIS`).

```
#2 X 2 matrix multiply over the ring Z_3
.model matmul
.inputs a11  a12  a21  a22
.inputs b11  b12  b21  b22
.outputs c11  c12  c21  c22
.mv a11,  a12,  a21,  a22  3
.mv b11,  b12,  b21,  b22  3
.mv c11,  c12,  c21,  c22  3
.table a11 a12 b11 b21 c11
0 0  - - 0
0 1  - - =b21
0 2  - 0 0
0 2  - 1 2
0 2  - 2 1
1 0  - - =b11
1 1  0 0 0
1 1  0 1 1
1 1  0 2 2
1 1  1 0 1
1 1  1 1 2
1 1  1 2 0
1 1  2 0 2
1 1  2 1 0
1 1  2 2 1
1 2  0 0 0
1 2  0 1 2
1 2  0 2 1
1 2  1 0 1
1 2  1 1 0
1 2  1 2 2
1 2  2 0 2
1 2  2 1 1
1 2  2 2 0
2 0  0 - 0
2 0  1 - 2
2 0  2 - 1
2 1  0 0 0
2 1  0 1 1
2 1  0 2 2
2 1  1 0 2
2 1  1 1 0
2 1  1 2 1
2 1  2 0 1
2 1  2 1 2
2 1  2 2 0
2 2  0 0 0
2 2  0 1 2
2 2  0 2 1
2 2  1 0 2
2 2  1 1 1
2 2  1 2 0
2 2  2 0 1
2 2  2 1 0
2 2  2 2 2
.table a11 a12 b12 b22 c12
0 0  - - 0
0 1  - - =b22
0 2  - 0 0
0 2  - 1 2
0 2  - 2 1
1 0  - - =b12
:
:
:
.table a21 a22 b11 b21 c21
0 0  - - 0
0 1  - - =b21
0 2  - 0 0
0 2  - 1 2
0 2  - 2 1
1 0  - - =b11
:
:
:
.table a21 a22 b12 b22 c22
0 0  - - 0
0 1  - - =b22
0 2  - 0 0
0 2  - 1 2
0 2  - 2 1
1 0  - - =b12
:
:
:
.end
```

The above example is stored in a file called `matmul`. We start `MVSIS` with the command `mvsis`. The following aliases are used:

```
rl                read_blifmv
saf               set autoexec print_stats -f
fs                fullsimp
pr                print_range
s                 simplify -t 2
pf                print_factor
pr                print_range
pio               print_io
pd                pair_decode
```

```
mvsis
UC Berkeley, MVSIS 1.2
changing to short-name mode
mvsis> rl matmul
mvsis> pr
{i}:    3
{j}:    3
{k}:    3
{l}:    3
a:      3
b:      3
```

```
c:      3
d:      3
e:      3
f:      3
g:      3
h:      3
mvsis> pio
primary inputs:   a b c d e f g h
primary outputs:  {i} {j} {k} {l}
mvsis> saf
matmul:  4 nodes,  4 POs,  128 cubes(sop),
               480 lits(sop),  216 lits(fact.)
mvsis> s
matmul:  4 nodes,  4 POs,  96 cubes(sop),
               320 lits(sop),  160 lits(fact.)
mvsis> pd 1
m{0} = a{0}e{2} + e{0}
m{1} = a{0}e{1}
m{3} = a{1}e{2} + a{2}e{1}
n{0} = a{0}f{2} + f{0}
n{1} = a{0}f{1}
n{3} = a{1}f{2} + a{2}f{1}
o{0} = e{0}c{2} + c{0}
o{1} = e{0}c{1}
o{3} = e{1}c{2} + e{2}c{1}
p{0} = f{0}c{2} + c{0}
p{1} = f{0}c{1}
p{3} = f{1}c{2} + f{2}c{1}
q{0} = b{0}g{2} + g{0}
q{1} = b{0}g{1}
q{3} = b{1}g{2} + b{2}g{1}
r{0} = b{0}h{2} + h{0}
r{1} = b{0}h{1}
r{3} = b{1}h{2} + b{2}h{1}
s{0} = g{0}d{2} + d{0}
s{1} = g{0}d{1}
s{3} = g{1}d{2} + g{2}d{1}
t{0} = h{0}d{2} + d{0}
t{1} = h{0}d{1}
t{3} = h{1}d{2} + h{2}d{1}
matmul:  12 nodes,  4 POs,  64 cubes(sop),
          184 lits(sop),  160 lits(fact.)
mvsis> s
matmul:  12 nodes,  4 POs,  56 cubes(sop),
          96 lits(sop),  96 lits(fact.)
mvsis> pf
{i}{1} = p{2}t{2} + p{1}t{0} + p{0}t{1}
{i}{2} = p{2}t{0} + p{1}t{1} + p{0}t{2}
{j}{1} = m{2}q{2} + m{1}q{0} + m{0}q{1}
{j}{2} = m{2}q{0} + m{1}q{1} + m{0}q{2}
{k}{1} = n{2}r{2} + n{1}r{0} + n{0}r{1}
{k}{2} = n{2}r{0} + n{1}r{1} + n{0}r{2}
{l}{1} = o{2}s{2} + o{1}s{0} + o{0}s{1}
{l}{2} = o{2}s{0} + o{1}s{1} + o{0}s{2}
m{0} = a{0} + f{0}
m{2} = a{2}f{1} + a{1}f{2}
n{0} = c{0} + e{0}
n{2} = c{2}e{1} + c{1}e{2}
o{0} = c{0} + f{0}
o{2} = c{2}f{1} + c{1}f{2}
p{0} = b{0} + g{0}
p{2} = b{2}g{1} + b{1}g{2}
q{0} = b{0} + h{0}
q{2} = b{2}h{1} + b{1}h{2}
r{0} = d{0} + g{0}
r{2} = d{2}g{1} + d{1}g{2}
s{0} = d{0} + h{0}
s{2} = d{2}h{1} + d{1}h{2}
t{0} = a{0} + e{0}
t{2} = a{2}e{1} + a{1}e{2}
matmul:  12 nodes,  4 POs,  56 cubes(sop),
          96 lits(sop),  96 lits(fact.)
mvsis> pr
{i}:    3
{j}:    3
{k}:    3
{l}:    3
a:      3
b:      3
c:      3
d:      3
e:      3
f:      3
g:      3
h:      3
m:      3
n:      3
o:      3
p:      3
q:      3
r:      3
s:      3
t:      3
```

7.2 Example 2

The second example is a 4-valued ALU and is one (`aluack.mv`) of the set of benchmarks distributed with `MVSIS` [2]. It also has a 4-valued control variable which determines which operation is done by the ALU.

The following additional aliases are used in this example. Note that `encode -i` does a binary encoding, but inserts encoders and decoders at the inputs and outputs to keep the inputs and outputs multi-valued. This allows a later verification (`vl` against the original network.

```
enm        encode -i
u          undo
m          merge
vl         validate -n 1000
fx         fast_extract
el         eliminate

mvsis> rl aluack.mv
alu:  7 nodes,  2 POs,  68 cubes(sop),
        140 lits(sop),  128 lits(fact.)
```

```
mvsis> fs
alu:  7 nodes,  2 POs,  48 cubes(sop),
        98 lits(sop),  96 lits(fact.)
mvsis> m
alu:  6 nodes,  2 POs,  41 cubes(sop),
        80 lits(sop),  80 lits(fact.)
mvsis> vl aluack.mv
Networks are combinationally equivalent
         according to simulation.
alu:  6 nodes,  2 POs,  41 cubes(sop),
        80 lits(sop),  80 lits(fact.)
mvsis> pr
{e}:    4
{f}:    2
a:      4
b:      4
c:      2
d:      4
h:      4
j:      4
k:      4
l:      9
alu:  6 nodes,  2 POs,  41 cubes(sop),
        80 lits(sop),  80 lits(fact.)
mvsis> pf
{e}{1} = d{3}j{1} + d{2}l{2,6} +
              d{1}h{1} + d{0}l{1,2}
{e}{2} = d{3}j{2} + d{2}l{4,7} +
              d{1}h{2} + d{0}l{3,4}
{e}{3} = l{5,6,7,8}(d{2}l{8} + d{0})
              + d{3}j{3} + d{1}h{3}
{f}{1} = c{1}j{0,3}k{0,1,3} +
             h{1,2,3}l{0,3,4,5,6,7,8}
h{1} = b{1}l{0,1,3,5} + l{7}
h{2} = b{2}l{0,1,3,5} + l{6}
h{3} = a{3}b{3}l{0,1,3,5}
j{1} = c{1}k{0} + c{0}k{1}
j{2} = c{1}k{1} + c{0}k{2}
j{3} = c{1}k{2} + c{0}k{3}
k{1} = l{2,6}
k{2} = h{3}l{5} + h{0,1}l{1,3,4}
k{3} = l{8}
l{0} = a{0}b{0}
l{1} = a{1}b{1}
l{2} = a{1}b{0} + a{0}b{1}
l{3} = a{2}b{2}
l{4} = a{2}b{0} + a{0}b{2}
l{5} = a{3}b{3}
l{6} = a{3}b{2} + a{2}b{3}
l{7} = a{3}b{1} + a{1}b{3}
alu:  6 nodes,  2 POs,  41 cubes(sop),
        80 lits(sop),  80 lits(fact.)
mvsis> enm
alu:  22 nodes,  2 POs,  87 cubes(sop),
        338 lits(sop),  199 lits(fact.)
mvsis> fs
alu:  14 nodes,  2 POs,  36 cubes(sop),
        81 lits(sop),  77 lits(fact.)
mvsis> fx
alu:  16 nodes,  2 POs,  37 cubes(sop),
        78 lits(sop),  75 lits(fact.)
mvsis> el 0
alu:  14 nodes,  2 POs,  35 cubes(sop),
         79 lits(sop),  74 lits(fact.)
mvsis> pr
{e}:    4
{f}:    2
a:      4
b:      4
c:      2
d:      4
d0:     2
e0:     2
h0:     2
j0:     2
k0:     2
l0:     2
m0:     2
n0:     2
o0:     2
p0:     2
q0:     2
w0:     2
alu:  14 nodes,  2 POs,  35 cubes(sop),
        79 lits(sop),  74 lits(fact.)
mvsis> pf
{e}{0} = d0{0}e0{0}
{e}{1} = d0{1}e0{1}
{e}{2} = d0{0}e0{1}
{f}{0} = l0{0}m0{0}(c{0} + h0{1}) + q0{1}
d0{1} = d{3}h0{1} + d{0,2}j0{1} + d{0,1}l0{1}
e0{1} = d{3}(k0{1}w0{0} + k0{0}w0{1}) +
             m0{1}(d{1,2}p0{0} + d{0,1}p0{1})
             + d{0,2}k0{1}l0{0}m0{0}
h0{1} = c{0}j0{1} + w0{1}
j0{1} = n0{1}o0{0}
k0{1} = j0{1}p0{0} + l0{1}p0{1} + m0{0}o0{1}
l0{1} = a{1,3}j0{0}
m0{1} = n0{0}o0{1} + p0{0}q0{0}
n0{1} = (a{0,2,3}b{0,1,2} + a{0,1,2}b{0,2,3})
        (a{1,2,3}b{0,1,3} + a{0,1,3}b{1,2,3})
o0{1} = q0{0}(a{1,3}b{1,3}n0{1} +
        a{0,2}b{0,2}) + a{1,2}n0{0}
p0{1} = a{1,2}b{1,2} + a{0,3}b{0,3}
        + b{0,2}o0{1}
q0{1} = a{0,1}b{0,1}
w0{1} = c{1}j0{0}
mvsis> vl aluack.mv
Networks are combinationally equivalent
                   according to simulation.
```

7.3 Example 3

The following example is an FSM where an EXDC is extracted due to the initial specification being incomplete.

```
.model lion9
.inputs i0 i1
.outputs ns o0
.mv i0 2
.mv i1 2
.mv o0 2
.mv ps, ns 9 st0 st1 st2 st3 st4 st5 st6 st7 st8
.latch ns ps
.reset ps
st0
.table i0 i1 ps -> ns o0
1 0 st0 st1 0
0 0 st0 st0 0
0 0 st1 st0 0
1 0 st1 st1 0
1 1 st1 st2 0
1 0 st2 st1 0
1 1 st2 st2 0
0 1 st2 st3 0
1 1 st3 st2 1
0 1 st3 st3 1
0 0 st3 st4 1
0 1 st4 st3 1
0 0 st4 st4 1
1 0 st4 st5 1
0 0 st5 st4 1
1 0 st5 st5 1
1 1 st5 st6 1
1 0 st6 st5 1
1 1 st6 st6 1
0 1 st6 st7 1
1 1 st7 st6 1
0 1 st7 st7 1
0 0 st7 st8 1
0 1 st8 st7 1
0 0 st8 st8 1
.end
```

The following external don't care network is produced, due to the incomplete specification of the above table:

```
.exdc
.inputs i0 i1 ps
.outputs ns o0
.mv ps 9
.table i0 i1 ps ->o0
.default 0
1 0  (3,{7-8})  1
0 1  ({0-1},5)  1
1 - 8 1
0 0  (2,6)  1
- 1 0 1
1 1 4 1
.table i0 i1 ps ->ns
.default 0
1 0  (3,{7-8})  1
0 1  ({0-1},5)  1
1 - 8 1
0 0  (2,6)  1
- 1 0 1
1 1 4 1
```

7.4 Example 4

The following example demonstrates `extract_seq_dc` that computes sequential don't cares (unreachable states) and merges them with the EXDC network. The example has multi-valued sequential don't cares (states $st5$ and $st6$ are unreachable from the initial state $st1$), to which additional don't cares are added after encoding because there is an unused code (101). The alias `wl` stands for `write_blifmv`.

```
mvsis> rl lb2.mv
node ns is incompletely specified
     (exdc extracted)
node o0 is incompletely specified
     (exdc extracted)
mvsis> wl
.
.
we omit the regular part of the network
.
.
.exdc
.inputs i0 i1 ps
.outputs ns o0
.mv ps 6 st1 st2 st3 st4 st5 st6
.table i0 i1 ps ->o0
.default 0
0 0 (st4,st5) 1
1 0  (st1,st3)  1
- 1 st2 1
0 1 st6 1
.table i0 i1 ps ->ns
.default 0
0 0 (st4,st5) 1
1 0  (st1,st3)  1
- 1 st2 1
0 1 st6 1
.end

mvsis> enm -v
node inf_ns is encoded as: value 0 - 001
   value 1 - 010   value 2 - 000
   value 3 - 011   value 4 - 100
   value 5 - 110
mvsis> extract_seq_dc
external don't care network has been created
mvsis> wl
.
.
.
.exdc
.inputs i0 i1 ps_b0 ps_b1 ps_b2
.outputs inf_ns_b2 o0 inf_ns_b1 inf_ns_b0 ns
.table ps_b0 ps_b1 ps_b2 i0 i1 ->o0
```

```
.default 0
1 - - - - 1
- 1 0 - 1 1
0 0 - 1 0 1
- 1 1 0 0 1
.table ps_b0 ->inf_ns_b2
.default 0
1 1
.table ps_b0 ->inf_ns_b1
.default 0
1 1
.table ps_b0 ->inf_ns_b0
.default 0
1 1
.table ps_b0 ps_b1 ps_b2 i0 i1 ->ns
.default 0
1 - - - - 1
- 1 0 - 1 1
0 0 - 1 0 1
- 1 1 0 0 1
.end
```

8 Comments

1. MVSIS can work correctly on non-deterministic networks, i.e. ones where some primary output has more than one value for some primary input minterm. If a network is non-deterministic, it can result in a new network that is not equivalent to the original but has a behavior that is **contained** in the original. The command verify checks that the containment is maintained.

2. MVSIS can be applied to binary files specified in BLIF. The results can be compared to those obtained by SIS. Currently, MVSIS compares favorably with SIS, when applied to the same binary file, both in terms of speed and quality of results. The quality is possibly due to some proceedures that are not part of SIS, such as fullsimp_complete which uses the complete set of don't cares to do the node minimizations. At the same time, it does "phase assignment" if the minimized complement has a simpler form.

3. MVSIS is available as executables running under either LINUX or WINDOWS [2].

4. A BLIF-MV file can be generated from Verilog using vl2mv which is available as part of VIS [3].

9 Conclusions and Further Remarks

The program MVSIS embodies a lot of work done by many people through the years on multi-valued synthesis. It can manipulate and optimize multi-valued multi-level networks and is the natural generalization of SIS which does binary network optimization. Our goal is to make MVSIS the system of choice for multi-level network optimization, be it binary or multi-valued, similar to how ESPRESSO-MV has replaced ESPRESSO-IIC in two-level logic minimization.

Applications of MVSIS are increasing and will increase further as this new capability is better understood and experimented with. Current developments include improvement of existing methods and experimentation with new ideas. Some of these come from the fact that the domain of optimization is expanded by opening up multi-valued possibilities. For example, we have discovered new binary methods by transforming to the multi-valued domain, performing some operations, and transforming back [13]. These possibly would not have been discovered by considering only the binary case.

Acknowledgements

We gratefully acknowledge the support of the SRC in funding this project under contract SRC-683.004. In addition, the logic synthesis class of Spring 1999 at Berkeley started MVSIS as a combined class project under the guidance of Subarnarekha Sinha, class TA. The class members were Yunjian Jiang, Niraj Shah, Scott Weber, Heloise Hse, Fernando De Bernardinis, David Chinnery, and Rupak Majumdar. The work of A. Mishchenko was supported by Intel, and T. Villa was partially supported by the GSRC.

References

[1] R. K. Brayton. Algebraic methods for multi-valued logic. Technical Report UCB/ERL M99/62, Electronics Research Laboratory, University of California, Berkeley, Dec. 1999.

[2] R. K. Brayton and et al. MVSIS. http://www-cad.eecs.berkeley.edu/Respep/Research/mvsis/.

[3] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa. VIS: A system for verification and synthesis. In *IEEE International Conference on Computer-Aided Verification*, 1996.

[4] C. Files and M. Perkowski. Multi-valued functional decomposition as a machine learning method. *IEEE International Symposium on Multi-Valued Logic*, pages 173–178, May 1998.

[5] M. Gao and R. K. Brayton. Semi-algebraic methods for multi-valued logic. In *Proc. of the Intl. Workshop on Logic Synthesis*, May. 2000.

[6] M. Gao and R. K. Brayton. Multi-valued multi-level network decomposition. In *Proc. of the Intl. Workshop on Logic Synthesis*, June 2001.

[7] T. Hanyu and M. Kameyama. A 200 mhz pipelined multiplier using 1.5v-supply multiple-valued mos current-mode

circuits with dual-rail source-coupled logic. *IEEE Journal of Solid-State Circuits*, 1995.

[8] J.-H. Jiang, Y. Jiang, and R. Brayton. An implicit method for multi-valued network encoding. In *International Workshop on Logic Synthesis*, June 2001.

[9] Y. Jiang and R. K. Brayton. Don't cares and multi-valued logic network minimization. In *Proc. of the Intl. Conf. on Computer-Aided Design*, Nov. 2000.

[10] Y. Jiang and R. K. Brayton. Logic optimization and code generation for embedded control applications. In *Proc. of the Intl. Symposium on Hardware/Software Co-Design*, Apr. 2001.

[11] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial hdl synthesis tools. In *International Symposium and Advanced Research in Asynchronous Circuits and Systems*, Apr. 2000.

[12] S. Minato. Fast generation of irredundant sum-of-products forms from binary decision diagrams. In *Proc. of SASIMI (Synthesis and Simulation Meeting and International Interchange)*, pages 64–73, 1992.

[13] A. Mishchenko and R. Brayton. Boolean paradigm in multi-valued logic synthesis. *To be submitted to International Workshop on Logic and Synthesis*, June 2002.

[14] A. Mishchenko and R. Brayton. Simplification of non-deterministic multi-valued networks. *To be submitted to International Workshop on Logic and Synthesis*, June 2002.

[15] A. Mishchenko and T. Sasao. Encoding of boolean functions and its application to lut cascade synthesis. *To be submitted to International Workshop on Logic and Synthesis*, June 2002.

[16] A. Mishchenko, B. Steinbach, and M. Perkowski. Bi-decomposition of multi-valued relations. In *International Workshop on Logic and Synthesis*, pages 35–40, June 2001.

[17] R. Rudell and A. Sangiovanni-Vincentelli. Exact Minimization of Multiple-Valued Functions. *IEEE Trans. Comput.-Aided Design Integrated Circuits*, 5:727–750, 1987.

[18] H. Savoj and R. K. Brayton. The Use of Observability and External Don't Cares for the Simplification of Multi-Level Networks. In *Proc. of the Design Automation Conf.*, pages 297–301, June 1990.

[19] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Laboratory, Univ. of California, Berkeley, CA 94720, May 1992.