MVSIS 2.0 User's Manual

Donald Chai, Jie-Hong Jiang, Yunjian Jiang, Yinghua Li, Alan Mishchenko, Robert Brayton

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley CA 94720
(mvsis-devel@ic.eecs.berkeley.edu)

Abstract

`MVSIS` is a program modeled after `SIS`, but the logic network it works on is such that all variables can be multi-valued, each with its own range. We include all the technology-independent transformations of `SIS` for combinational logic synthesis as well as transformations specific to multi-valued nodes such as `merge, pair_decode, encode, print_range, reset_default`. `MVSIS` has been made to have the look and feel of `SIS`. `MVSIS` can read and write `BLIF-MV` and `BLIF` files with the `read_blif_mv` command as well as binary PLAs in the ".type fd" format.

MVSIS 2.0 is a major re-write of MVSIS 1.0. The goals of MVSIS 2.0 were to significantly speed up the processing of logic, and to provide a clean source code for the first time. It is expected that future logic synthesis research will benefit by implementing new ideas in MVSIS, even for binary logic. In the long run, MVSIS will be a complete replacement of SIS, but with the added capability of handling multi-valued non-deterministic networks. This user's manual describes the essential features included in the MVSIS 2.0 release.

1 Introduction

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

1. Logic synthesis for multi-valued hardware devices.
2. Initial manipulation of a hardware description before it is encoded into binary and processed by standard binary logic synthesis programs; 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. Strong logic synthesis transformations can be applied to compilers aimed at embedded applications.
4. Asynchronous synthesis

We have developed and included techniques for combinational optimization of MV networks. Like `SIS` [1, 2], `MVSIS` is an interactive tool, and has been made to have the look and feel of `SIS`. When applied to purely binary networks, it behaves almost exactly like the technology independent part of `SIS` but is faster. In the sequel, the main components of `MVSIS 2.0` are described, including the input specification, the MV-transformations and descriptions of commands.

The underlying data-structures used in logic synthesis have been completely revised in `MVSIS 2.0` for maximum efficiency. At this point in time, on binary files, `MVSIS 2.0` is about 3-5 times faster than SIS, uses much

less memory, and can be applied more robustly to much larger designs. The data-structures have been designed with numerous APIs[1] to make the writing and debugging of new packages much easier. Almost all of the capability of `MVSIS 2.0` is in the area of technology independent optimization which has many new features. Many of the algorithms have both a SOP-based version and a BDD based version. This allows dynamically choosing the version that is faster under the given local circumstances. Node minimization is extended with a BDD approach, based on Minato's algorithm, to deal with very large two-level functions if ESPRESSO fails. New algorithms have been developed to derive and use complete output observability relations of a node in the network (complete flexibility). For algebraic methods, a completely new binary version of "fast_extract" has been written which can handle any size of a problem in almost no time. A multi-valued version of this uses the so-called CST transformation which maps the problem into the binary domain where the new fast binary method is used. Multi-valued algorithms such as factor, `decomp` and `fx`, are supported using this transformation. Tests show that little optimality is lost, while the algorithms are much faster than the corresponding MV algorithms in earlier versions of MVSIS.

In the MV domain, non-deterministic networks are properly handled according to a theory that has been developed [3]. There are several algorithms which convert to and from the MV domain; `pair` `decode` `merge` and `encode`. These allow exploration of optimization potential in the MV domain before all signals are encoded into binary.

In this manual we give an overview of `MVSIS 2.0` and review the ideas behind multi-valued logic synthesis. We do not give all possible ways to call a `MVSIS 2.0` command with all of its options, but for each command, there is a short description of the options allowed. This can be seen by invoking `<command> -h`. Refer to the Programmer's Manual for details on the data-structures.

2 Design Specification

2.1 `BLIF-MV` Description Format

An MV circuit can be input to `MVSIS` as a netlist of MV-nodes (command: `read_blif_mv`). We use a simple subset of the `BLIF-MV` [4] format that was used in `VIS` to specify the design. Such `BLIF-MV` files can be generated by the Verilog front-end to `VIS (vl2mv)` or can be written out by `VIS`. Binary networks specified in `BLIF` (the format most commonly used in `SIS`) can also be read in using the same command (`read_blif_mv`). After a design specification is read in, it is converted into an MV-network, a design representation used within `MVSIS`. An MV-network is a network of nodes; each node represents an MV-relation with a single multi-valued output. The functions associated with the onsets of the values (i-sets), i.e. the binary-output function which is 1 if the relation can take on value *i*, of a node are stored in SOP or MDD form. There is one MV variable associated with the output of each node. An edge connects node *i* to node *j* if any of the i-sets at *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 in our representations, each variable can have a separate range of its own, including two values. All ranges are represented by the sets $\{0, 1, \ldots, n_i - 1\}$, $n_i > 1$.

`MVSIS 2.0` supports sequential MV-networks with multi-valued latches but does no sequential optimization. A latch, as with any other variable in the network, can be multi-valued. In the `BLIF-MV` file, a latch can be specified using construct `.latch`, with the initial state specified using construct `.reset`. We only support `BLIF-MV` files with constant initial state values. All latch inputs are treated as primary outputs and all latch outputs as primary inputs. The set of all inputs is denoted as CI (combinational inputs) and the set of all outputs is denoted as CO (combinational outputs).

[1] Application Programming Interfaces

The initial specification as well as the current network may have non-deterministic relations at the nodes. This may result in one or more of the primary outputs to be non-deterministic as functions of the primary inputs. The result of synthesis may be a subset of the initial relation specified. This is defined to be valid as long as the network is well-defined, i.e. for every input minterm, there exists at least one allowed value for each output.

2.2 External Specification

For multi-valued nodes, if a `.default` value is given, its i-set is defined to be the complement of the union of all the other i-sets. Thus the node is completely specified in the sense that it has an allowed output value for each input minterm. The node may still be non-deterministic if the table specifying the non-default values has a minterm with more than one allowed output value. For a table without the `.default` value, if the sum-of-products specification 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. The omission of a "default" in the input file allows the specification of external don't cares. In some applications, such as data mining, the otherwise specified minterms may consume much space.

In the execution of some commands, it is important to re-derive the flexibility allowed from an external specification file (an MV network). We assume either that the initial file is the spec or that the initial spec is specified in a file indicated with the `.spec` option. An example of this use is the `mfs` command. If the option `-s` is given it will read in the "spec" file and use this to derive the flexibility that can be used during `mfs` optimization. If no `-s` option is given, `mfs` will use the current network as the spec. On writing out the current file, the `BLIF-MV` file will include a `.spec` notation. Currently, only `mfs` has the ability to use an external specification file.

3 Combinational Optimization

3.1 Node Simplification

One of the ways to simplify the logic i-sets (one for each output value) at an MV-node is with the `simplify` command which uses the two-level logic minimizer `ESPRESSO-MV`. The objective of a general two-level logic minimizer is to find a logic representation with a minimum number of implicants (cubes) and literals while preserving functionality. Satisfiability don't cares from the local fanins and subset support variables are used in the minimization unless the `-d` option is used. After simplication, the i-sets are replaced with simplified versions if the new functions have been improved according to the cost function in use. Currently, there are 3 cost functions that can be used which can be controlled by the global `set` command. If `set cost 0`, the total number of cubes in all relations is used, 1 uses the number of literals, and 2 uses the number of literals in the factored forms.

For each node, after an initial file is read in, one of the i-sets is selected as the default value. For example, for a binary output function, the onset is usually the primary value (non-default) and the offset is the default value. The concept of a default i-set is useful because it is never looked at (and is not part of the cost function evaluation) unless a command requires it. For example, if the output of a binary function is used in the complemented form in a fanout and the node is eliminated, then the complement must be computed to effect the elimination. The values of the nodes and statistics of the network are based only on the non-default i-sets. However, there is one command `reset_default` which looks at each node and chooses a default value for it based on the cost of the node. For example, if the cost function is the number of cubes, `reset_default` will cause all value functions to be minimized, and the default value will be chosen as the one whose function has the most cubes.

In the future, there may be more complex cost functions depending on the target of the application, e.g., the number of nodes, the number of values, the number of fanins or some physical information.

One of the strongest kinds of node simplification that can be performed on a network, is implemented using the `fullsimp` (alias `fs`) command. To perform this function on a multi-level MV-network, a don't care set is automatically generated. Subsets of the satisfiability and observability don't care sets (SDC and ODC respectively) are used. The notion of compatible observability don't cares (CODC) used in `SIS` [5] has been generalized to take MV-nodes into account [6]. Given these, MV-image computation techniques are used to map them to the local fanin space of each node. An additional SDC of those nodes in the network whose support is a subset of the support of the node being simplified is also added to the local don't care set thus derived. This allows a form of Boolean substitution when `fs` is executed. The `-r` option suppresses Boolean re-substitution. Each node is then simplified by `ESPRESSO-MV` using this local don't care set and the option `-m nocomp` (only in the binary case).

During any of the above forms of simplification, if it is estimated that simplification will take too long, the node will be minimized with a simpler form of minimization or left unchanged. The complexity of an `ESPRESSO-MV` session is estimated by the number of cubes in the onset and don't care set, and the number of fanin variables. If this is too large, `ESPRESSO-MV` is not called. There is also a timeout for the `fullsimp` and `simplify` commands, controlled by the `-t` option. The specified time (in integer seconds) is shared among three time consuming computations; CODC computation, image computation, and `ESPRESSO-MV` minimization. If any one of these takes longer than the allocated time, the simplification for that node is terminated and only the local SDC is used for the node minimization.

A more powerful node simplification method [7], called `mfs` performs the same overall 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[2] between the node's fanins and its output (generally multi-valued). This relation gives all allowed combinations of inputs and outputs of the node, i.e. when they appear at the node, will not violate the overall network behavior allowed at the primary outputs. It is a *complete* description of a node's flexibility (hence called the complete flexibility of a node (CF)).

2. The flexibility computation and node simplification are interleaved. The reason for this is that the complete flexibility at one node is not compatible with that of another node; thus a node must be optimized immediately after the flexibility is computed. 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[3]. 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 is a useful optimization, since its use 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 or in most binary synthesis programs.

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.

[2] 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.

[3] Note that because of the use of the default value, this can't happen in the binary case

If the `mfs` command is given with the ”-k” option and a node name, it will simplify the node and display the Karnaugh map of both the initial relation at the node and the derived complete flexibility relation, which is usually non-deterministic. Executing this again will display the node relation obtained after minimizing the complete flexibility relation.

An example is the following:

```
mvsis 115> mfs -k m1
Original MV Relation of Node <m1>.
m \ l
       0          1          2          3          4
  +----------+----------+----------+----------+----------+
0 | --2----- | ---3---- | ---3---- | -1------ | ---3---- |
  +----------+----------+----------+----------+----------+
1 | ------6- | ---3---- | -------7 | -----5-- | -------7 |
  +----------+----------+----------+----------+----------+
2 | ------6- | ---3---- | -------7 | -----5-- | -------7 |
  +----------+----------+----------+----------+----------+
3 | ----4--- | 0------- | ----4--- | ----4--- | ----4--- |
  +----------+----------+----------+----------+----------+
Derived MV Relation of Node <m1>.
m \ l
       0          1          2          3          4
  +----------+----------+----------+----------+----------+
0 | 012345-- | 012345-- | 01234567 | 012345-- | -1-3-5-- |
  +----------+----------+----------+----------+----------+
1 | ------6- | -1-3-5-- | -------7 | -1-3-5-7 | ------67 |
  +----------+----------+----------+----------+----------+
2 | ------6- | -1-3-5-- | -1-3-5-7 | -1-3-5-7 | ------67 |
  +----------+----------+----------+----------+----------+
3 | 0-2-4--- | 0-2-4--- | 0-2-4-6- | ----4--- | 012345-- |
  +----------+----------+----------+----------+----------+
pal3x:  ci/co = 6/1  lat = 0  nd = 11  cube = 45  ff-lit = 140  lev = 5
mvsis 116>
```

Here the notation is that the set of numbers given at a minterm location are the output values allowed. For example `-1-3-5--` denotes that only the output values `1,3,5` are allowed, while `0,2,4,6,7` are not allowed. The entrie with `01234567` most likely comes from SDCs. `-1-3-5--` is an example of a partial care. Note that if only don’t cares would be used, there is only one minterm in the don’t care set. In contrast, there are 16 minterms with partial cares.

A new capability in `MVSIS 2.0` is the use of windowing. Currently, only the ”mfs” command uses this, e.g. with the command `mfs -i n -o m`. Here n specifies the number of levels in the TFI to include in the window and m the number of levels in the TFO to include. All nodes, such that all paths from the selected node to a node is less or equal to $m(n)$ in the TFO (TFI), are included in the window. In addition, any side node which is on any path from a node in the TFI_n to a node in the TFO_m is included. The corresponding window $W_{n,m}(v)$ is then extracted from the network and the node v is minimized using the window as its own specification. The resulting change at node v is then put back in the network. Then ”mfs” moves on to the next node, its window extracted and the node minimized. The use of windows allows the ”mfs” command to be applied to arbitrarily large networks, since the BDDs associated with a window network can always be built (if the parameters for the window n, m are limited).

Part of the optimization improvements of `MVSIS 2.0` is due to the use of much superior BDD methods which in certain cases have been specialized for the types of operations found in MVSIS and in general in logic synthesis.

3.2 Algebraic Methods

An important step in network optimization, uses algebraic methods for extracting new nodes representing logic functions that are common factors of other nodes. We developed and implemented in `MVSIS 1.0` new algebraic techniques for MV-logic [8, 9, 10] 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 an SOP form. For descriptions of these, refer to the previous releases of MVSIS.

However, a technique called Co-Singleton Transform (CST) was developed since then [11]. This uses a special encoding of MV into binary to map the network into a binary one. Then the algebraic binary operations are performed (using fast algorithms[4]) to obtain a new network. Finally, the network in mapped back into an MV network. It has been shown that the use of the CST leads to little loss in optimality when used in an overall optimization script, while the results are obtained significantly faster[5].

The relevant commands and brief descriptions of their abilities are listed below.

1. The command `fxu` looks at all the nodes in the network and tries to extract good common factors (two-cube divisors and cubes) and creates 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.

2. The command `decomp` does a complete factoring of each node, but instead of creating a factored form for each, decomposes the node according to its factorization. Thus more intermediate nodes are produced this way. Such intermediate nodes may not have been produced by `fxu`, so there is a possibility of finding different (better?) factors. After this, elimination can be done to clean up the network, possibly followed by `simplify, fs, mfs` to look for Boolean substitutions.

3. The command `strash` is like `decomp` but it replaces the current network by a functionally equivalent network composed of only ANDs and inverters (which are collapsed into the fanout AND-gates, unless a fanout is a CO). This network is constructed using a structural hashing algorithm similar to [12]. If the original network is multi-valued and non-deterministic, the resulting network is equivalent to the original one in terms of SS-behavior [3].

3.3 Network Manipulations

1. **Collapsing** converts the entire multi-level network so that the i-sets 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 combinational outputs (COs). Our version of collapse [13] is based on building the MDDs of the outputs, and deriving an ISOP [14] for each value. In many binary examples it will be noticed that this is much faster than the `SIS` version. Generally, this is very fast if the MDDs can be built efficiently. In addition the use of Minato's ISOPs gives a result that is already partially minimized.

2. **Elimination** of a node consists of substituting the node relation into all the node's fanouts and then removing the node from the network. The elimination command requires a value above which a node will not be eliminated if its "value" exceeds this threshhold. Elimination can be done either by manipulating cubes or BDDs. The method used is automatically chosen in MVSIS depending on the the method which is estimated to be the most efficient.

[4] Orginally these were imported from SIS, but in `MVSIS 2.0` all these operations have been improved considerably, e.g. by 5X, so no external calls to SIS are required

[5] These operations are much faster than in SIS, due to their improved binary versions.

3. **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. The new i-sets are the intersections of the corresponding i-sets of the combinations. For example, if `merge x y z` is given for 3 binary variables, a new variable w with 8 values will be constructed, where e.g. $w^{\{5\}} = x\bar{y}z$ 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. Since some new i-sets may be empty, the number of i-sets created may be much fewer. In addition, if a pair of values always appears together in all the fanouts, then their functions will be combined (unioned) into a single i-set. Merging is one of the methods for creating intermediate MV nodes. Note that node extraction and decomposition only create binary output nodes, since these methods are based on AND/OR factoring.

4. **Pair decode** is an automatic process (in contrast to `merge`) which looks around for variables to merge. These include the combinational inputs (CIs). It automatically finds promising candidates and evaluates them for their potential savings by examining the i-sets that would be created, as well as the effect on the fanouts' i-sets. The `pair_decode` operation is similar to input pairing that has been done for PLA synthesis in the past. It is another method which can create intermediate MV nodes.

5. **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, excluding primary inputs and outputs. Encoding is performed from combinational outputs to combinational inputs in topological order. This order can be reversed by the -d option. There are three encoding methods specified by the `-m` option:

   - `in` Input encoding considers the optimization of a multi-value node's fanouts. As few binary codes are used as possible such that more satisfiability don't cares can be used to optimize its fanout nodes.
   - `out` This considers the optimization of the new binary nodes, which replace the original multi-valued node.
   - `inout` automatically switches between input and output encoding for a multi-valued node, depending on the number of fanouts, the number of binary nodes, and the number of excessive binary codes.

4 Verification

MV-networks are verified in `MVSIS 2.0` by formal methods. For this, the global i-sets are computed for each output using an MDD representation (like symbolic simulation) and compares the MDD structures with the network's external specification. Since sequential optimization is not performed, this comparison is purely combinational. Since `MVSIS 2.0` supports optimization of non-deterministic networks [7], formal verification checks for containment instead of equivalence. The command `verify` can take a file as an argument. In this case, the current network is compared to the network in the file. For convenience, `verify` without any argument will check verification against the last file read in by a read command. If containment is not present, an error vector will be printed out.

5 Sequential Optimization

Currently, unlike SIS, all optimizations are performed on the combinational part of the sequential network. At present no sequantial optimization operations are included in `MVSIS 2.0`.

6 Other Commands and Options

6.1 Iteration

Several commands have the option to apply the command a given number *n* times, or until a fixed point, i.e. no change occurs in the network. The command structure is, for example `fs -i n` where *n* is the iteration count. If no `-i` option is given, iteration to a fixed point is implied.

6.2 Printing, reading and writing

There are several write commands, e.g. `write_blif_mv`, `write_blif`, and several read commands, e.g. `read_blif_mv` and `read_pla`. `read_blif_mv` automatically detects if a file is `BLIF` or `BLIF-MV` and acts appropriately.

To view the results at any stage, there are several print commands which print to the console. `print` prints for all nodes, the SOP form of each i-set. If the `-d` option is given it will also display the default i-sets. `print_factor` prints, for all nodes, the factored form of each non-default i-set. Each of these can take, as an argument, a list of names of nodes to be printed. In general, *, like in `SIS`, stands for a list of all nodes in the network.

`print_range` prints out the size of the range for each variable; `print_value` the value of each node (according to the current cost function); `print_stats` the statistics of the network in terms of the network name, the number of combinational inputs/outputs, the number of nodes, the number of cubes, and the number of literals in the factored and the number of levels in the networkps. `print_stats -s` prints out a number of a number of other statistics. `print_io` prints the inputs and outputs of the network. If a list of nodes is given, it prints out the fanins and fanouts of each node in the list. Command `print_domi` computes and prints the immediate dominators of the requested nodes, or all nodes in the network.

Sometimes, in order to view an output or factorization better, it is useful to change the names of the variables to short names using the command `chng_name`. It is a toggle between short names and the original names. Associated is a command `reset_names` which resets the naming of short names for the variables so that all variables appear in lexicographic order with no gaps in the naming. Inputs are named first, $\{a, b, c, \ldots\}$, then outputs, and finally intermediate nodes.

6.3 History and Undo

A new history capability has been added. The user will note that each command line is given a number which denotes the state of the network at a result of the last command. If a command can't have changed the state, the number is not increased after this command. `MVSIS 2.0` saves a number of backup states (copies) of the networks which is specified by the global set command `set savesteps n` (default is $n = 1$). As before `undo` toggles between the current state and the previous state. For larger jumps, `snatch <line number>` will revert back the the specified state if it has been saved.

6.4 Miscellaneous Commands

1. **rename** This command permanently changes the names of all nodes (including CIs and COs) to short names which are constructed using lower-case alphanumeric characters only. Note that, after running this command, the network cannot be verified against the original network, because verification is based on comparing the functionality of two networks with identical CI/CO names. Command "rename" is useful in two cases: (1) to permanently get rid of very long node names, which for large networks may consume considerable memory and are not efficient for accessing the nodes through the hash table hashing node names into node pointers,

and (2) to disguise industrial netlists before releasing them as public domain benchmarks (hopefully, this will encourage more companies to do so:). Note that this command is different form **chng_name** which just toggles between short names and long (original) names.

2. **reset_name** This command is the same as in `SIS`. It compacts the short names of the nodes. However, unlike `SIS` the short names are not stored in the nodes. They are derived on the fly from the node IDs (unique integer numbers assigned to each node). Therefore, "reset_name" in MVSIS compacts the node IDs, which leads to shortening of the short names.

3. **print_nd** Prints the list of all non-deterministic nodes in the current network.

4. **default** This is a very useful command. When run without command line arguments, it assigns the default value for the nodes whenever possible. When run with the argument "-d", it does the reverse, unassigns, or deletes, the default value. In this case, if the default value was previously set at a node, the corresponding default i-set is computed and inserted into the node's representation, so that the default value is no longer used.

5. **dize** Determinizes the current network if it was non-deterministic. Does not change the network if it is deterministic. The determinization performed is a brute-force one. The i-sets of each node are considered in their numeric order and each of the following i-sets is "sharped" with the OR of the i-sets considered before.

6. **free** This command is used to free one of the two functional representation at each node of the current network. To free the MV SOP representation, type "free cvr". To free the MDD MV relation representation, type "free mvr". Each representation is stored at a node using lazy evaluation, i.e. they are only computed and stored when required by a command. If a particular representation has been computed, it is stored at the node. If another command changes the node's functionality, the stored representation is invalidated. `free` forces the invalidation of the named type of representation.

7. **window** This command is used to test the efficiency of windowing for a particular circuit. A window is defined by giving a list of nodes, called the window core, (typically, a window core may be composed of one node only) and two parameters, which specify the number of levels of limited TFI and TFO to include. The window with these parameters is constructed by including into it the nodes of the current network, which fall into at least one of the following four categories: (1) the nodes of the window core; (2) the nodes in the TFI of the window core, such that **all** paths from the core to them is less than the given parameter; (3) the nodes in the TFO of the window core, such that **all** the path from the core to them is less than the given parameter; (4) all the nodes that are in the TFO of nodes of category (2) and in the TFI of nodes in the category (3).

   Typing "window -a" prints the statistics for all windows with the specified parameters centered in each node of the current network. If switch "-a" is not used, but instead a list of node names is provided on the command line, the list of nodes is assumed to represent a window core. The window centered at this core is computed and written into a BLIF or BLIF-MV file (depending on whether the network in binary or MV). The name of the output file is derived from the network's model name A, the name B of the first node in the core, and the total number N of nodes in the resulting window: "A_B_N.blif" or "A_B_N.mv".

8. The command `club` groups nodes into a set of disjoint clusters and collapses all nodes in the same cluster. The default approach is to compute the prime dominators for each node (a dominator that is not dominated by any other node), and collapse all nodes into its prime dominators. This tends to give results that lie between those of eliminate and collapse. An alternative approach (method greedy) visits all nodes from primary inputs

to primary outputs in a levelized order, and heuristically merges nodes into a cluster such that the sharing of fanin and fanouts is maximized within a cluster. The size of a cluster is limited by parameters settable at the command line (number of input/output bits and total number of literals).

Several experimental commands, that are not already included in the manual and are not listed above, are prefixed with an underscore ("_") and hidden from the user. However, typing "help -a" will list all commands including the hidden ones.

6.5 Setting global parameters

The `set` command sets various global parameters, which control the transformations. With no argument, `set` prints out the current values of the global parameters.

1. `cost` controls the type of cost function to be used in the evaluation of the value of a node. `set cost 0` uses the number of cubes as the cost function; `set cost 1` uses the number of literals in the factored form SOP, and `set cost 2` uses the number of literals in the factored form expressions.

2. `namemode` can be set to 0 to use long names default and to 1 to use short names.

3. `autoexec` can be given a command which will automatically execute after each command line is executed. A typical use is `set autoexec print_stats -c`, which will print out the statistics (including literals in the factored forms) of the network after each transformation.

4. `alias` is like `set`. It is used to create nick-names for various commands. For example `alias ps print_stats` can be used to print out the stats of the network with the single command `ps`. `alias` with no arguments will print out a list of all aliases defined so far. A typical set of aliases is incorporated in the file `master.mvsisrc`, which is executed when `MVSIS` is started. Such a file must be in the same directory as `mvsis`. `savesteps` has been discussed above in the "history" section.

6.6 Help and Scripts

1. `help` prints the set of commands available. Any command with the option "-h" or "-?" will print a detailed description of the command.

2. `source` reads and executes commands from a file (a `.script` file).

7 Caveats and Comments

1. In general, some `MVSIS` commands do not have time-out filters yet. Some computations which use the complement operation or BDDs may blow up sometimes. We have filters in `fullsimp`, `mfs` and `eliminate` to try to control this, but it is possible that an operation will not terminiate. In that case it is necessary to use `CTRL_C` to terminate MVSIS.

2. `MVSIS 2.0` is for the first time being made available in source code. It will compile under LINUX and WINDOWS.

3. `MVSIS 2.0` almost always works correctly on non-deterministic networks, i.e. ones where some primary output has more than one value for some primary input minterm. Like the use of external don't cares in the binary case, the new network may not be equivalent to the original but should have a behavior that is **contained** in the original. The command `verify` checks that the containment is maintained. In rare cases if a network has internal ND nodes, the resulting network may not verify. The source of this is known and a theory exists about how to fix this [3]. However, in this release of `MVSIS 2.0`, since such occurrences are rare we have chosen to not modify this version. Such a non-verification can only happen in the non-binary case. Therefore for MV examples, it is important to do a final verification to determine that the resulting network is indeed correct. In general, we have found that although in practice it is possible that a network can get out of spec, many of the MV optimizations in `MVSIS 2.0` have the potential for self-correction and this seems to explain why non-verification is rare.

4. `MVSIS 2.0` can be applied to files specified in `BLIF` or `BLIF-MV` using the command `read_blif_mv` which automatically detects the type of file to be read. Results on binary examples can be compared to those obtained by `SIS`. Currently, `MVSIS 2.0` compares favorably with `SIS` and is much faster. Since "phase assignment" is automatic in MVSIS, often the results are better. For large examples, MVSIS can easily outperform SIS, since fullsimp in SIS will time-out and provide no optimization, whereas MVSIS can use windowing. In addition, the `mfs` command provides more flexibility for minimization.

5. `MVSIS 2.0` is available in source code running under either Linux or Windows [15].

6. A `BLIF-MV` file can be generated from Verilog using `vl2mv` which is available as part of the `VIS` [16] release. Alternatively, it can be generated from Esterel programs, using the translator `dc2mv`, which converts declarative code (DC) formats produced by the Esterel compiler to `BLIF-MV`.

8 Conclusions and Further Remarks

The new version of MVSIS `MVSIS 2.0` embodies a lot of effort by many people over many years working on both binary and multi-valued synthesis. `MVSIS 2.0` can manipulate and optimize multi-valued multi-level non-deterministic networks and is the natural generalization of `SIS` which does binary network optimization. Our goal has been to make `MVSIS 2.0` the system of choice for multi-level network optimization, be it binary or multi-valued. A similar occurrence happened when `ESPRESSO-MV` replaced `ESPRESSO-IIC` in two-level logic minimization. In this latest release, for the technology independent optimizations, the goal has been achieved.

Applications of true multi-valued logic are increasing and should increase further as this new capability is better understood and experimented with. Future developments include many improvements, already known for existing methods. In addition, there are many new ideas in logic that need experimentation. 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 several new binary methods by transforming to the multi-valued domain, performing some operations, and transforming back [17]. These possibly would not have been imagined by considering only the binary case.

Much of the future interest use of MVSIS will be on binary applications. In order to include the many binary logic synthesis options of SIS (e.g. technology mapping), we plan to release a version which allows both MVSIS and SIS to co-exist in memory. Operations (commands) which unique SIS will be automatically intercepted and executed by converting the internal data-structures to those of SIS, applying the appropriate command, and mapping the result back to the MVSIS data-structures. The overhead for doing this will be minimal and unnoticed since MVSIS will make up for it by its the greater speed on the commands it has. As more algorithms are re-coded for the MVSIS data-structures, SIS should slowly migrate out of existence.

Acknowledgements

We gratefully acknowledge the support of the SRC in funding this project, under contract SRC 683.004, over many years. Although this support ends in June 2003, MVSIS development will continue under separate funding from severals compainies whose main interest is in binary logic synthesis. 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 has been supported by Intel under a separate contract. Other contributors to MVSIS have been supported by the California Micro program and our industrial sponsors, Cadence, Fujitsu, and Synplicity.

References

[1] 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," Tech. Rep. UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[2] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," in *Proc. of the Intl. Conf. on Computer Design*, pp. 328–333, Oct. 1992.

[3] A. Mishchenko and R. Brayton, "A theory of non-deterministic networks," in *International Workshop on Logic and Synthesis*, May 2003.

[4] R. K. Brayton, M. Chiodo, R. Hojati, T. Kam, K. Kodandapani, R. P. Kurshan, S. Malik, A. L. Sangiovanni-Vincentelli, E. M. Sentovich, T. Shiple, K. J. Singh, and H.-Y. Wang, "BLIF-MV: An Interchange Format for Design Verification and Synthesis," Tech. Rep. UCB/ERL M91/97, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, Nov. 1991.

[5] 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.*, pp. 297–301, June 1990.

[6] 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.

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

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

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

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

[11] J.-H. Jiang, A. Mishchenko, and R. Brayton, "Reducing multi-valued algebraic operations to binary," in *Proc. of DATE*, Mar. 2003.

[12] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification," in *Trans. CAD*, pp. 1377–1394, Dec. 2002.

[13] A. Mishchenko, R. Brayton, and T. Sasao, "Exploring multi-valued minimization using binary methods," in *International Workshop on Logic and Synthesis*, May 2003.

[14] 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)*, pp. 64–73, 1992.

[15] R. K. Brayton and et al., "MVSIS." `http://www- cad.eecs.berkeley.edu/mvsis/`.

[16] 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.

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