MVSIS*

M. Gao, J.-H. Jiang, Y. Jiang, Y. Li, S. Sinha, and R. Brayton
EECS Dept. University of California, Berkeley CA 94720

Send comments to: mvsis-devel@ic.eecs.berkeley.edu
Free download from:
http://www-cad.eecs.berkeley.edu/Respep/Research/mvsis/

Abstract

`MVSIS` is a program modeled after `SIS`, but it works on logic networks where all variables can be multi-valued, each with its own range. It includes all the technology-independent transformations of `SIS` for combinational logic synthesis plus many transformations specific to multi-valued nodes. Input and output in `MVSIS` is provided through `BLIF-MV` files. It can also read `BLIF` files. It is available at the web site given above.

1 Introduction

Although much work has been done on multiple-valued optimization [1, 2, 3, 4, 5], there has been no efficient package for the general multi-level optimization of multi-valued networks. This is primarily because there has not existed a complete set of efficient tools for MV-network manipulation. `MVSIS` encorporates new methods which have been specially developed and tuned for `MVSIS` to provide a complete package for efficient manipulation of MV-networks [6, 7, 8, 9]. It is competitive with `SIS` in terms of speed and quality of results.

`MVSIS` optimizes multiple-valued (MV) combinational networks. It is an interactive tool, and has been modeled after `SIS` [10] as much as possible. On purely binary networks, it behaves almost exactly like the technology independent part of `SIS` for combinational optimization.

Multi-level multi-valued (MV) logic synthesis can have possibly many applications and in the future we intend to explore several of these using `MVSIS`. Applications include:

1. Logic synthesis for multi-valued hardware devices [11].
2. Initial manipulation of a hardware description before it is encoded [6] 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; software lends itself naturally to the evaluation of multi-valued variables in a single cycle [12]. Strong logic synthesis transformations can be applied to compilers aimed at embedded applications.
4. Asynchronous synthesis [13].

We describe the input specification, the MV-transformations and special commands, verification by simulation, followed by a few examples illustrating the use of `MVSIS` in the design process. We emphasize those aspects where the multi-valued nature leads to differences from `SIS`.

2 Design Specification

An MV circuit can be input to `MVSIS` as a netlist of MV-nodes (command: `read_blifmv`). We use a simple subset of `BLIF-MV` [14], used in `VIS`, to specify the design. Such `BLIF-MV` files can be generated by the Verilog front-end to `VIS (vl2mv)` or be written out by `VIS`. Binary networks can also be read (`read_blif`). Internally, the design representation is a network of MV nodes; each node represents an MV-function with a single multi-valued output. An important distinction with some other MV methods, is that each `MVSIS` variable can have a separate range of its own, including two values. The range for variable y_k is represented by the set $P_k = \{0, 1, \ldots, p_k - 1\}$. The function whose onset gives the set of minterms for which $f_k = i$ (the function at node k equals the value i) is called the *i-set* of function f_k and is stored in SOP form. In the 2-valued case, the 0-set corresponds to the off-set and the 1-set to the on-set. One MV variable y_k is associated with the output of node k. An edge connects k to j if any of the i-sets of j depends explicitly on y_k. The network has a set of primary inputs (all of which may be multi-valued) and a set of primary output nodes.

*We gratefully acknowledge funding from SRC under contract 683.004. The logic synthesis class of Spring 1999 at Berkeley started `MVSIS` as a combined class project (class members: Yunjian Jiang, Niraj Shah, Scott Weber, Heloise Hse, Fernando DiBenederos, David Chinnery, and Rupak Majumdar; class TA: Subarna Sinha).

3 Combinational Optimization

3.1 Node Simplification

The i-sets (one for each output value) at an MV-node can be simplified with the `simplify` or `fs` commands which use ESPRESSO-MV [15]. `simplify` uses satisfiability don't cares (SDC) from the local fanins and, unless the `-d` option is used, also subset support variables are used. After simplication, the SOPs for the i-sets are replaced if the new ones are simpler.

For each node, one i-set is designated as the default and by definition is the complement of the other i-sets. In the binary case, the onset is usually the primary value and the offset, the default value. The default i-set is computed only if a command requires it. The values of the nodes and statistics of the network are based only on the non-default i-sets. However, one command `reset_default` looks at each node and chooses a new default based on the costs of the i-sets. For example, if the cost function is the number of cubes, `reset_default` will cause the i-sets to be minimized, and the default values chosen as the i-sets with the most cubes.

Currently, two cost functions can be used, one which counts the number of literals in the factored form and the other the number of cubes. Both only look at non-default values.

The strongest kind of node simplification is `fullsimp` (alias `fs`). Compatible observability don't cares (CODC), generalized to take MV-nodes into account [9], and MV-image computations [9] are used to generate a don't care set in the local space of the node k being simplified. An SDC of other nodes whose support is a subset of the support of k is added (*this allows a form of Boolean substitution when* `fs` *is executed*). Each of the node's i-sets is then simplified by `ESPRESSO-MV` using this local don't care set.

3.2 Kernel and Cube Extraction

An important step in network optimization uses algebraic methods for extracting new nodes that are common divisors of other nodes. We have developed new algebraic techniques for MV-logic [7] which treat binary and multi-valued variables uniformly. They include methods for finding common sub-expressions, semi-algebraic division, decomposing a multi-valued network, and factoring a SOP form. The relevant commands and brief descriptions of their abilities are listed below.

1. The command `fx` extracts good common divisors and creates new nodes in the network, re-expressing other nodes in terms of these. It has two options, `-q` and `-g`. The first generates candidate two-cube divisors by making each pair of cubes in a node i-set cube-free. These 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 are also kept if it is a two-cube expression. The divisor with the largest value is then extracted as a new node and substituted into all functions where applicable. For efficiency, the divisor table is only incrementally updated after each substitution.

   The `-g` option can generate additional divisors. It first generates the same divisors as `fx -q`, but additionally generates a set of candidate double cube divisors, one for each function, by factoring each node in the network. It then divides these candidate divisors, used in the factorization, into all other nodes to compute their values. This option can be time consuming if there are i-sets with a large number of cubes. This is special to the multi-valued case, since in the binary case, it would not generate any new divisors.

2. The command `decomp` does a complete multi-valued factoring of the i-sets of each node and decomposes the nodes according to these factorizations. After this, `resub`, using multi-valued algebraic division [7], can be executed to eliminate duplicate factors.

3. Algebraic substitution of one node into another is performed in `MVSIS` using `resub [-d] [node-list]`. All i-sets of the divisor node are tried; the default i-set is also tried unless the `-d` option is given. However, division into the default i-sets of other nodes is not done, since those i-sets can be obtained by complementation. `resub` uses the new methods of "exact" semi-algebraic division, developed for multi-valued logic [7, 8]. There are two modes; if the divisor is a two-cube divisor, then a fast method based on matching is used; otherwise, a slower branch and bound method is used (called the satisfiability-matrix method). Although in theory all pairs of nodes must be examined during `resub`, similar to `SIS` (but not the same), there are effective filters to determine if a pair could never cause an algebraic substitution.

3.3 Other Network Manipulations

The commands `collapse, undo` and `eliminate` operate exactly as in `SIS`. Other commands specific to MV networks are:

1. `merge` takes a list of nodes and forces a merge of them into a single multi-valued node. 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 i-sets are merged into a single i-set. If no list is given, merge automatically looks for likely candidates and merges them if a gain in the value for the network can be obtained. This may result in several additional multi-valued nodes through multiple merges. In case `merge` is given a list, the nodes will be merged regardless of the gain, provided the merging does not introduce cycles. Cycles will happen if there is a path from some node A to some node B in the list, which passes through a node not in the list.

2. `encode` tries to find a good binary encoding for each multi-valued variable in the network, including primary inputs and outputs [6]. At the end, each signal is encoded as a binary signal, including primary inputs and outputs. Then a binary file can be written (but as a `blif-mv file`). An option (`-i`) puts encoders and decoders at the input and outputs, and keeps the original multi-valued inputs and outputs. The internal signals are binary and the network between the encoders and decoders is exactly the same as if the `-i` option was not used. This encoded circuit can be validated against the original file. A good operational rule is to first encode a file with the `-i` option, validate the circuit, and then call `encode` to create the purely binary file. `encode` without the `-i` option will simply strip off the front-end and back-end encoders and decoders.

3. `pair_decode` does "bit-pairing" to create new multi-valued nodes. It looks for a "best" pair of signals to pair together. Then it creates a new node (if its value is greater than a specified threshold) with i-sets equal to all the decodes of the pair. For example, for two binary signals, a 4-valued node is created and algebraically substituted into nodes which depend on at least one of these decodes. Finally, any set of values of the new node, which always appear together in the fanouts, is merged into a single value of the new node. After this step, `simplify` should be executed to effect full substitution.

4. `eliminate_part` is like eliminate, except it works only on multi-valued output nodes and can eliminate some of the i-sets for the nodes. A specified threshold controls which i-sets are to be eliminated. Values are given to each i-set heuristically and the i-sets are ordered. To see the accumulated value of the i-sets, the command `print_part_value` shows the ordering of the i-sets and the accumulated values from least to greatest. For example, for a particular node `m` it might print out the following,

```
m:   (110)   12 30 50 68 88 112 136 160 184
 (6 3 0 7 8 5 2 1 4)
```

The name `m` is first followed by the normal value of the node (110), then a vector of the accumulated i-set values $12, 30, \ldots$, where 12 is the value of the 6-set, 30 is 12 plus the value of the 3-set, etc. If the command `eliminate_part 30` is given, `MVSIS` will merge the 6-set and the 3-set, and then select a new default value, since the newly merged (6,3) i-set may be the largest.

3.4 Iteration, Timeouts and Filters

Several commands have the option (unlike `SIS`) to apply the command `n` times, or until no change occurs in the network. These commands are `eliminate, pair_decode, fx` and `fullsimp`. The command structure is, for example `eliminate -i n <threshold>` where `n` is the iteration count. If no `-i` option is given, iteration to a fixed point is implied.

Many commands have a timeout option `-t`. For example, `pair_decode -t 1 10` will cause a timeout after 1 second has been used for finding a best pair. When it times out, the best pair seen so far is selected and if its value is greater than 10 will cause a new node to be produced. Commands with the `-t n` option are `fullsimp, simplify, pair_decode, collapse, fx`. Other commands have internal timeouts while `print_factor` and `decomp` have a timeout controlled by a global parameter, `time_limit`.

Several commands have filters on the number of cubes in an i-set. If the number exceeds a threshold, the operation is skipped on that node. These commands include `reset_default, fx, eliminate` and `merge`.

3.5 Printing, Reading and Writing

There is one write command, `write_blifmv`, and two read commands, `read_blifmv` and `read_blif`. The latter reads in ordinary `BLIF` files. Currently (but coming soon), `MVSIS` is not able to read `BLIF` or `BLIV-MV` files with registers.

To view the results at any stage, print commands can be used. `print` prints (to the console) the SOP form of each i-set (including the default if the option `-d` is given) of each node in the network. `print_factor` prints the factored form of each i-set (excluding the default). Each can take a list of names of nodes to be printed.

`print_range` prints the range magnitudes of each variable; `print_value` the value of each node (according to the current cost function); `print_stats` the statistics of the network (network name, the number of primary outputs, the number of nodes, the number of cubes, and the number of literals in the cubes). `print_stats -f` also prints the number of literals in the factored forms of the non-default i-sets. `print_io` prints the inputs and outputs of the network. If a list is given, it prints the fanins and fanouts of each node in the list.

3.6 Setting global parameters

The `set` command controls the global parameters. With no argument, `set` prints out the current values of all global parameters.

1. `cost` controls the type of cost function to be used evaluating nodes. `set cost 0` uses the number of cubes as the cost function; `set cost 1` the number of literals in the factored form.

2. `time_limit`, controls the maximum time (in seconds) spent in factoring or decomposing a single i-set. Since MV-algebraic factoring may expensive, this timeout is especially useful when the factoring is done only to estimate the value of a node or to produce a readable output.

3. `el_limit` controls when a node will be eliminated. For each fanout, the number of cubes that will result after elimination is estimated. If this exceeds the maximum of `el_limit` and twice the largest cover of any i-set, the elimination of the current node is aborted. A similar control is in `SIS`.

A typical set of aliases is incorporated in the file `master.mvsisrc`, which is executed when `MVSIS` is started. Some are illustrated in the Appendix.

4 Verification

MV-networks are only incompletely "verified" in `MVSIS` by simulation. `validate -n N` verifies the combinational equivalence of two networks by simulating the networks on N random vectors, and comparing the outputs. `gen_vec -n N <file>` generates N random input vectors appropriate for the ranges of the primary inputs, and writes them `file`. This can be used by `simulate <file>`, which produces simulation results at the primary outputs. Formal MDD-based verification will be implemented soon in `MVSIS`.

5 Caveats

1. `MVSIS` only works correctly on deterministic networks. Note that, unlike Boolean networks, an MV-network can be nondeterministic. A sufficient test for determinism is that for each function, its i-sets do not intersect. (Boolean networks have, by definition, the 0-set of any node equal to the complement of the 1-set). However, it is quite common that i-sets intersect for multi-level MV-networks because of SDC and ODC don't cares. `MVSIS` does a quick but incomplete check for non-determinism when a new circuit is read in.

2. `MVSIS` applied to binary files can be compared to results obtained by `SIS`. In general, `SIS` is bit faster but `MVSIS` is more rugged because of its more extensive use of timeouts. The quality of results produced by `SIS` and `MVSIS` are comparable.

3. `MVSIS` is available only as a binary executable for Redhat Linux 6.1 (or Linux kernel 2.2.x and glibc 2.1).

4. `MVSIS` will accept registers in the near future.

5. External don't cares will be allowed soon in `MVSIS` using an external don't care network.

References

[1] E. L. Post, "Introduction to a general theory of elementary propositions," in *Amer. J. Math.*, June 1921.

[2] J. C. Muzio and D. M. Miller, "On the minimization of many-valued functions," in *Proc. 9th Int. Symp. Multiple-Valued Logic*, June 1979.

[3] S. L. Hurst, "Multiple-Valued Logic - its status and its future," in *IEEE Trans. om Computers*, June 1984.

[4] J. C. Muzio and T. C. Wesselkamper, *Multiple-valued switching theory.* Bristol: Hilger, 1986.

[5] R. K. Brayton and S. P. Khatri, "Multi-valued logic synthesis," in *Proc. 12th Int. Conf. on VLSI Design*, Jan. 1999.

[6] J.-H. Jiang, Y. Jiang, and R. Brayton, "An Implicit Method for Multi-Valued Network Encoding," in *Proc. of IWLS*, June 2001.

[7] M. Gao and R. K. Brayton, "Semi-Algebraic Methods for Multi-Valued Logic," in *Proc. of the IWLS*, May 2000.

[8] M. Gao and R. Brayton, "Multi-Valued Multi-Level Network Decomposition," in *Proc. of IWLS*, June 2001.

[9] Y. Jiang and R. K. Brayton, "Don't Cares and Multi-Valued Logic Network Minimization," in *Proc. of ICCAD*, Nov. 2000.

[10] 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.

[11] T. H. T. and M. Kameyama, "A 200 MHz pipelined multiplier using 1.5 V-supply multiple-valued MOS current-mode circuits with dual-rail source-coupled logic," in *IEEE Journal of Solid-State Circuits*, 1995.

[12] Y. Jiang and R. Brayton, "Logic optimization and code generation for embedded control applications," in *CODES*, Apr. 2001.

[13] private communication, Theseus Logic Inc., http://www.theseus.com.

[14] 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.

[15] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," in *IEEE Trans. on Computer-Aided Design*, June 1987.

Appendix: Example Sessions

We illustrate MVSIS with two examples that are included in the MVSIS distribution.

Example 1

The following aliases are used:

```
gx   fx -g
rl   read_blifmv
saf  set autoexec print_stats -f
fsfullsimp
rsd  reset_default
pr   print_range
ssimplify -t 2
pf   print_factor
mmerge
pr   print_range
pio  print_io
el   eliminate
rsn  reset_name
pd   pair_decode
```

MVSIS is started with the command mvsis. Using the .mvsisrc file, the cost function is set to 1, i.e. cost is the number of literals in the factored forms and short names mode is toggled.

```
UC Berkeley, MVSIS 0.0.1
   (compiled 13-Feb-01 at 5:54 PM)
changing to short-name mode
mvsis> rl matmul-c
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> s
mvsis> saf
matmul:  4 nodes,  4 POs,  96 cubes(sop),
320 lits(sop),  160 lits(fact.)
mvsis> rsd
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> rsn
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}
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
```

Example 2

The following additional aliases are used:

```
enm  encode -i
u undo
m  merge
vl  validate -n 1000
```

```
UC Berkeley, MVSIS 0.0.1(
   (compiled 13-Feb-01 at 5:54 PM)
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> rsd
```

```
alu:  7 nodes,  2 POs,  49 cubes(sop),
 98 lits(sop),  97 lits(fact.)
mvsis> u
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> fs
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.
mvsis> pr
{e}:  4 {f}:  2 a:  4 b:  4 c:  2 d:  4 h:  4
j:  4 k:  4 l:  9
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}
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> gx
alu:  16 nodes,  2 POs,  37 cubes(sop),
 79 lits(sop),  77 lits(fact.)
mvsis> fs
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
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.
```

Note in this example, after `enm`, several operations were performed on the circuit like `fs`. Even though all the internal nodes are binary valued, since the primary inputs are still multi-valued, some SOP forms are binary output multi-valued input, so the final circuit is still not truly binary. To get a truly binary circuit, `encode` should be used. Then further network manipulations can be done without causing a multi-valued circuit (as long as `merge` and `pair_decode` are not called). For example, the included script `mvsis.scriptb` can be sourced to further simplify the network without creating any multi-valued nodes.