MVSIS v1.1 Manual

Jie-Hong Jiang, Yunjian Jiang, Yinghua Li, Alan Mishchenko*, Subarna Sinha
Tiziano Villa**, Robert Brayton

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley CA 94720
* Portland State University, Portland OR
** Parades, Rome Italy
(mvsis-devel@ic.eecs.berkeley.edu)

Abstract

This users manual describes the essential new features included in the MVSIS v1.1 release.

1 Introduction

Many new features have been added to MVSIS in the areas of technology independent and technology dependent optimizations. Node minimization is extended with an ISOP-based approach to deal with very large two-level functions if ESPRESSO fails. New algorithms have been developed to derive complete output observability relations of a node in the network. For algebraic methods, "EBD" mapping converts multi-valued problems into binary ones, which are then manipulated using binary algorithms, such as factor, decomp and fx, ported from SIS. The results are then converted back to MV. Tests show that no optimality is lost but the algorithms are much faster than the corresponding MV algorithms in MVSIS

As one application of MV logic, a new command `bidecomp` decomposes an MV network into a network of primitive MV devices consisting of MIN, MAX, and literal gates. Potentially, this could be used in data mining applications and circuit implementation using current-mode devices.

Another application is software synthesis for embedded control systems. Given an MV network, command `gen_c` produces a software implementation, i.e. a low-level C program that simulates the sequential network. This uses an extended version of the BLIF-MV format, with support for data-paths. Although the data-path portion of the network cannot by optimized, the data-flow information is used in minimizing the control logic.

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

Table 1: Node types in a control data network

node types	operation	input	output	example
control	logical	MV	MV	$a\{0\}b\{1,2\}+a\{1\}$
data	arithmetic	data	data	`x + y`
multiplexer	assignment	MV/data	data	$c\{0\}\mathtt{x}+c\{1\}\mathtt{y}$
predicate	predicate	data	MV	`x > y`

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.

When targeted for embedded system applications, the pure MV logic network is extended to have abstract data variables. These variables can be thought of as carrying an infinite range of values, which in the actual implementation can be mapped to any arbitrary type. Three additional types of nodes are supported for functions involving data variables: expressions, multiplexers and predicates. These are classified according to their input and output variables types, as shown in Table 1. In the examples in the table, *a*, *b*, *c* are multi-valued control variables, and *x*, *y* are data variables.

A multiplexer is defined as $f = f(y_c, y_0, \ldots, y_{n-1})$, where y_c is a MV-variable with n values, y_i, ($i \in [0, n-1]$) are data inputs. The output f is assigned to y_i if $y_c = i$. The data computation contained in predicate nodes and expression nodes are currently modeled as uninterpreted strings, but they must be arithmetic as definable by the semantics of the C language. As a result, these nodes cannot be reasoned about or simulated inside the MVSIS environment (only the control nodes can). The behavior of the entire network can be simulated only by generating C programs and then running the compiled program separately.

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

Figure 1 shows an example of a control-data network with two latches, where bold wires indicate data variables. These networks can be derived from Esterel programs, by the following tool flow (for an Esterel program named simple.strl):

```
% esterel -causal simple.strl
% areaopt simple
% blifsc -ctbl simple.ctbl simple.opt.blif > simple.opt.sc
% scdc simple.opt.sc
% dc2mv -p simple.opt.dc > simple.mv
```

`esterel, areaopt, blifsc, scdc` are tools released with the Esterel compiler, which performs analysis and optimization on the Esterel program and produces Declarative Code (DC). `dc2mv` is a parser released with MVSIS, which converts the DC format into an extended BLIF-MV format. `dc2mv` works on a subset of the DC format; for details refer to [1]. The DC format is a shared format among a number of synchronous languages, e.g. Lustre, Signal, Argos and StateChart. This makes MVSIS a common back-end optimization and mapping tool for synchronous applications developed with all these languages.

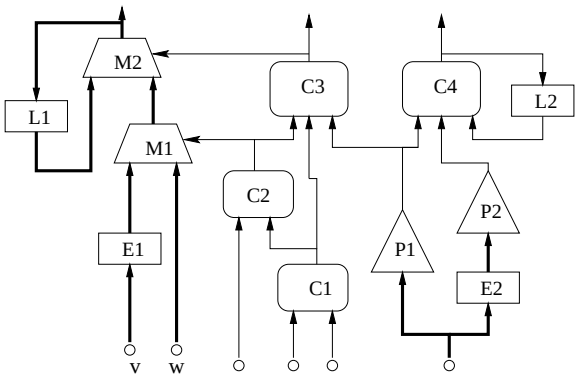

Figure 1: Control-data network

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` [2], 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 [3] 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 examined 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` is the direct generalization of the one in `SIS`. The notion of compatible observability don't cares (CODC) used in `SIS` [4] has been generalized to take MV-nodes into account [5]. Given these, MV-image computation techniques are used to map them to the local space of each node. In addition, an 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 [6], called `complete_simplify` 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

[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.

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 at one node is not made 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. 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.

When a data-path is present, the observability don't cares are also extended to take into account the data flow [7] in command `fullsimp -d`. Essentially, each node (both control and data) is computed for its output observability, which is then passed along to its own inputs. Additional ODCs are computed for inputs of a multiplexer node. This has been shown to be effective when the control portion and data portion of the network are highly intertwined and dependent on each other.

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. 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 release manual.

In addition, a technique called Encoding Binary Decoding(EBD) mapping is developed, which uses a special encoding into binary to map the network into a binary one. Then the algebraic binary operations are performed (using fast algorithms imported from SIS) to obtain a new network. Finally, the network in mapped back into an MV network. These lead to the new commands `ebd_fx, ebd_decomp` which can be used to replace `fx, decomp` respectively. It has been shown that the use of the EBD commands leads to no loss in optimality when used in an overall optimization script, and the results are obtained much faster.

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 `collapse_global` is based on building the MDDs of the outputs, and deriving an ISOP [3] 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. The new i-set is the intersection of the corresponding i-sets of the combination. 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 (unioned) 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 its termination, each signal has been encoded as a set of binary signals. 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.

   Two encoding schemes have been developed. The first one (command `encode`) 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. The second (command `encode2`) starts from the inputs and proceed topologically to the output. Each node is encoded using its local function as described in [11].

4 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 (it is like symbolic simulation) and compares the MDD structures; 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 [6]. In these cases formal verification checks for containment instead of equivalence.

Sometimes it is important to know if a network is initially non-deterministic. `MVSIS` has a built-in but incomplete test `qcheck` 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.

5 Technology Mapping

5.1 Bi-decomposition

This 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 [12]. Both the incompleteness of the initial specification and the flexibilities generated in the decomposition process are exploited. Bi-decomposition can be viewed as a kind of technology mapping step resulting in a network of multi-valued primitives analogous to NAND and NOR gate decomposition used in binary synthesis. This method is particularly suited for data mining because the maximum and minimum relations are easily understood by humans.

5.2 Code Generation

Here we focus on applications in control intensive embedded systems, where systems are designed using high-level synchronous languages like Esterel [13]. This high-level designs can in turn be compiled into a network of extended finite state machines (EFSMs) represented in terms of MV control networks (with data-path extension). We then synthesize efficient software implementations in C (command `gen_c`). This problem highly resembles the classical logic simulation problem with the same goal of high speed evaluation of logic networks. However, the tighter constraints of embedded systems in both code size and response time makes it a harder problem.

Code generation involves generating efficient evaluation code for MV logic functions, based on using possibly different representation with MDDs, SOPs or look-up-tables. It also involves scheduling of the nodes in the network with the goal of minimizing the evaluation effort [14].

6 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` using `read_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 sometimes improved, possibly due to some procedures that are not part of SIS, such as `complete_simplify` 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 [15].

4. A `BLIF-MV` file can be generated from Verilog using `vl2mv` which is available as part of `VIS` [16]. 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`.

5. The following commands can not be performed on networks that contain data nodes:

```
ebd_decomp, ebd_fx, sis_eliminate
isop, bidecomp, collapse_global, encode, encode2
verify, complete_simplify
```

7 Conclusions and Further Remarks

The program `MVSIS` embodies a lot of effort done by many people through the years working 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 [17]. These possibly would not have been imagined by considering only the binary case.

Appendix: Extended BLIF-MV

The new BLIF-MV file format accepted by MVSIS v1.1 extends the one defined in MVSIS v1.0 manual with support for data-path, as defined below.

- **Abstract data** variables are specified using the .n construct:

```
.n local_time
```

- **Multiplexer** node assumes the first input is an MV variable, whose value range is at least the number of data input variables:

```
.mv time_zone 3
.n pacific_standard_time
.n beijing_time
.n london_time
.mux time_zone pacific_standard_time beijing_time london_time -> local_time
0 - - -  =pacific_standard_time
1 - - -  =beijing_time
2 - - -  =london_time
```

- **Expression** node takes an un-interpreted string (contained in double quote marks “”) as input, which is assumed to conform to the semantics of the C language, and produces a data output.

```
.data pacific_standard_time -> pacific_daylight_time
"pacific_standard_time + 1"
```

- **Predicate** node is the same as an expression node, except that the output is a control variable (MV).

```
.data pacific_standard_time -> at_night
"(pacific_standard_time > 18) && (pacific_standard_time < 6)"
```

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] Y. Jiang and R. K. Brayton, *MVSIS Code Generation Manual*, June 2002.

[2] R. Rudell and A. Sangiovanni-Vincentelli, "Exact Minimization of Multiple-Valued Functions," *IEEE Trans. Computer-Aided Design*, vol. 5, pp. 727–750, 1987.

[3] 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.

[4] 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.

[5] 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.

[6] A. Mishchenko and R. Brayton, "Simplification of non-deterministic multi-valued networks," in *Proc. of the Intl. Workshop on Logic Synthesis*, June 2002.

[7] Y. Jiang and R. K. Brayton, "Don't care computation in minimizing extended finite state machines with presburger arithmetic," in *Proc. of the Intl. Workshop on Logic Synthesis*, Jun. 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] A. Mishchenko and T. Sasao, "Encoding of boolean functions and its application to lut cascade synthesis," in *Proc. of the Intl. Workshop on Logic Synthesis*, June 2002.

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

[13] G. Berry and G. Gonthier, "The Esterel synchronous programming language: Design, semantics, implementation," *Science of Computer Programming*, 1992.

[14] Y. Jiang and R. K. Brayton, "Software synthesis from synchronous specifications using logic simulation techniques," in *Proc. of the Design Automation Conf.*, June 2002.

[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," in *Proc. of the Intl. Workshop on Logic Synthesis*, June 2002.