A Boolean Paradigm in Multi-Valued Logic Synthesis

Alan Mishchenko
Department of ECE
Portland State University
alanmi@ece.pdx.edu

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

Abstract

Optimization algorithms used in binary multi-level logic synthesis, such as network simplification, logic extraction, and resubstitution, have been treated independently and did not share computational procedures. Using multi-valued logic synthesis, some common conceptual and computational cores underlying these algorithms can be identified.

We present an overview of a Boolean paradigm[1] in multi-valued logic synthesis. The Boolean algorithms are generalizations of the usual ones and can replace these and the traditional algebraic algorithms, offering improved trade-offs between computation speed and optimization quality.

1 Introduction

Traditional technology-independent logic synthesis flow exemplified by SIS [18] consists of transformations applied to a multi-level binary logic network. The transformations attempt to improve the sum of literal counts in factored forms of binary nodes as well as other cost functions.

Multi-level logic synthesis research of the 1980's [3][5] led to so-called *algebraic* optimization algorithms. These modify the SOP representation of the nodes in a restricted way, without using the identities of Boolean algebra, such as $a \wedge a = a$ and $a \wedge \bar{a} = 0$, nor do they give due to network structure. Algebraic transformations work well for many practical circuits but they are inferior in optimization quality to Boolean transformations, which use the complete scope of functional properties.

Improvements in algebraic techniques [8][10] and their Boolean counterparts [9][20] were developed later but their use has been restricted to experimental tools. This restriction occurs because of their inherent complexity and/or lack of good functional representations. This paper presents a common view of a set of network optimization algorithms used in multi-level logic synthesis. The distinctive aspects of this approach are:

- It exposes some relations among the algorithms (node simplification using internal don't-cares, logic extraction, decomposition-mapping, resubstitution, and encoding).
- The algorithms considered are essentially Boolean; they can make use of complete don't-cares[2] generated during synthesis and could improve optimization quality.
- A common computational core of the algorithms is outlined and expressed in terms of Boolean satisfiability.

In our experience, formulating optimization problems in terms of multi-valued logic [4] has helped in exposing relations among algorithms. Although the main algorithms shown in Figure 1 are usually considered essentially independent in the binary domain, it may be useful to view these in the multi-valued logic domain with a common theory and computational core (Boolean satisfiability).

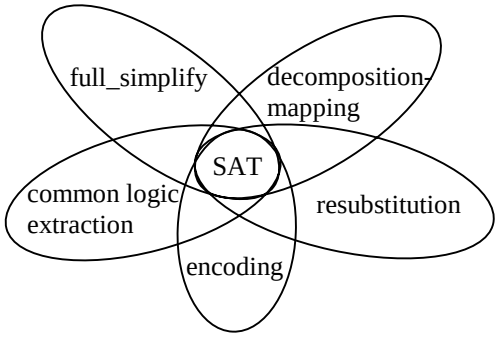

Figure 1. Relationship among algorithms.

Exploring common features of these algorithms may open new avenues for efficient implementation. The practical complexity of these procedures depends on the use of don't-cares and the type of optimization performed. Ignoring don't-cares, limits optimization to the current functions at the nodes but enables efficient heuristic solutions by reducing most of the algorithms from general Boolean satisfiability to graph coloring or other more efficient

[1] In this paper, Boolean is contrasted with algebraic, and binary is contrasted with multi-valued. As a result, it is possible to speak of a Boolean paradigm in multi-valued logic synthesis. These concepts are explained in the introduction.

[2] Don't cares in the MV domain are generalized to "partial cares" which are a form of non-determinism.

methods. The new paradigm involves exploration of efficient runtime trade-offs between constraint-based SAT or branch-and-bound procedures versus present methods.

A serious limitation of Boolean methods is their greater computational complexity resulting in excessive runtime. This limitation can be addressed potentially by developing efficient partitioning algorithms and applying Boolean methods to individual parts in an iterative fashion. Experiments show that the loss of quality due to partitioning for some optimization problems does not exceed a few percent [7]. In other problems, controlling the size of partitions may be used to trade optimization quality for runtime.

The paper is organized as follows. Section 2 elaborates on the use of internal don't-cares. Section 3 discusses some optimization algorithms in more detail. Section 4 relates one of these, minimum disjoint decomposition of an MV relation, to Boolean satisfiability. Section 5 concludes and outlines future work.

For definitions of multi-valued functions, multi-valued networks, and the flexibility derived for a node in the network, please refer to [12].

2 Internal Don't-Cares in Logic Synthesis

The optimization algorithm, *full_simplify*, in SIS [18] uses a subset of Observability Don't-Cares (ODCs) called Compatible Observability Don't-Cares (CODCs) [17]. These together with Satisfiability Don't-Cares (SDCs) are computed for each node in a multi-level network and used to simplify the node representations.

Recently it was shown that Complete ODCs [12] could be used to improve optimization results. Complete ODCs are not compatible, meaning that the set of don't-cares computed for a node is valid only as long as other nodes remain unchanged. Experimental results [12] show that the reasonable amount of computational overhead for Complete ODCs is justified by the improved quality of node simplification.

Generally, there are two cases, (1) when no flexibility is used, and (2) when either CODCs or Complete ODCs are used. In the first case, which is typically used, computational complexity is lower and it is often possible to achieve good results by applying a greedy algorithm. The second case constitutes the main topic of this paper.

3 Optimization Algorithms

This section presents Boolean formulations of several network optimization algorithms. The order, in which they are considered, is chosen to facilitate presentation of a common theme.

3.1 Network Optimization using Complete Flexibility at a Node

An initial network is shown in Figure 2 (left). Global functions $F_1(x),\ldots, F_k(x)$ are expressed in terms of the primary input (PI) variables x. Node η is the node under consideration.

The flexibility at η is computed by cutting the network at the output of η and introducing a new primary input variable z to replace the output of η. Then the global functions $F^z_1(x,z),\ldots, F^z_k(x,z)$ in terms the PIs and z are computed for the new network, as shown in Figure 2 (right).

The condition that the original specification should contain the behavior of the network modified by introducing additional variable z is transformed into relation $R^\eta(x,z)$ representing the complete flexibility at node η in terms of the PIs. This relation is then imaged into the local input space of η to derive the complete flexibility used for node optimization. In general, in the MV domain, this relation is non-deterministic and contains partial cares, meaning that under a given minterm, the output value of the node can take any of the allowed subset of values. Details of this algorithm are presented in [12].

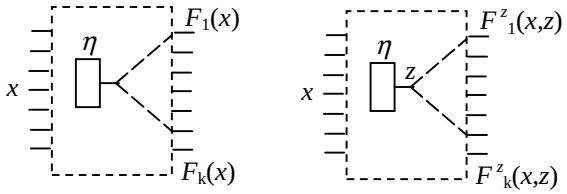

Figure 2. Computation of complete flexibility at a node.

The remaining algorithms of encoding decomposition extraction and substitution will be treated using this flexibility.

3.2 Partial Encoding

The problem of partial encoding is to reduce the number of values of a given multi-valued output node in a network by using already existing nodes in the network, or by using new but simple nodes.

This problem can be solved using a branch-and-bound algorithm coupled with a specialized decision diagram operators [13]. A partial encoding is optimal if as many coding functions as possible can be used which belong to functions already present in the network and hence can be replaced by a single "wire".

Instead of encoding the current function at a node, better results can be obtained by partially encoding the complete flexibility relation at a node shown in Figure 3 (left). Generally, a partial encoding splits relation R into two

relations R_1 and R_2 shown in Figure 3 (right) such that the total number of binary code-bits does not increase[3]:

$$\lceil \log_2 |v_2| \rceil + \lceil \log_2 |v_1| \rceil \leq \lceil \log_2 |v| \rceil.$$

When the number of binary code-bits remains the same, a tie-breaking condition is that the total number of values does not increase: $|v_2| + |v_1| \leq |v|$. We call such encodings *value reducing partial encodings.*

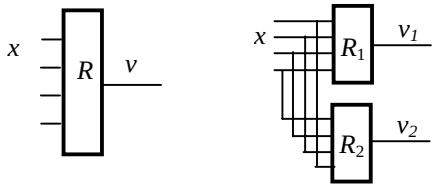

Figure 3. Encoding for value reduction.

In MV network optimization, we are interested in finding a partial encoding that leads to a simplified representation of blocks R_1 and R_2 compared to the original block R.

One approach to this problem restricts the MV relation R_2 to a wire or to an MV *cube* depending on a subset of the x variables. Thus we are looking for a new *simple* function to be used in the partial encoding. The characteristic function of the relation, $R_1(x,p)$, with respect to all possible cubes selected for R_2 can be computed using one binary variable p_{ki} for each value appearing in the cube variables; p_{ki} is set to 1 if value i is present in the literal of variable k of the cube. A selection of values for the p_{ki} provides a particular cube for R_2. Note that R_1 is derived from R, the complete flexibility at the node.

The resulting partial encoding problem is thus parameterized using variables p and the complete flexibility. A solution is found as a set of assignments to p, which reduces the number of values in relations R_1.

3.3 Non-Disjoint Decomposition and Encoding

The MV node to be decomposed is represented by its MV complete flexibility relation $F(x)$, shown in Figure 4 (left). A bound set x_B is selected and a disjoint decomposition is performed, yielding a new block, B_1, depending on x_B , and function F_1 as shown in Figure 4 (center). The number of values v_1 of block B_1 is simply assumed to be the product of the number of values of all variables in x_B[4]. Next, we compute the complete flexibility relation $R^{B1}(x,z)$ of block B_1 in the new network, where B_1 and F_1 have replaced F. This flexibility depends on input variables x_B and an output variable z having the $|v_1|$ values of B_1.

Now, we apply partial encoding to the flexibility relation $R^{B1}(x,z)$. If a value-reducing encoding exists using a code function that is a fanin variable x_C, a partial encoding is found. Variable x_C remains in the support of B_2 but also becomes part of the free set x_F, as shown in Figure 4 (right), leading to a non-disjoint decomposition.

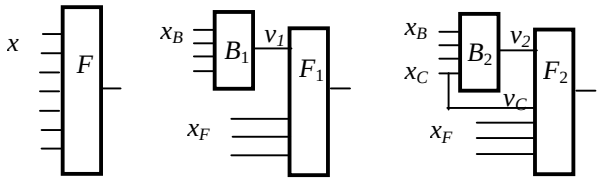

Figure 4. Decomposition and encoding.

Because this transformation is value-reducing, the number of values of $|v_2|$ of block B_2 satisfies $\lceil \log_2 |v_2| \rceil + \lceil \log_2 |v_C| \rceil \leq \lceil \log_2 |v_1| \rceil$ or $|v_2| + |v_C| \leq |v_1|$.

3.4 Resubstitution

The partial encoding technique can be used to search for a decomposition with an arbitrary function $G(x_B)$ instead of the particular case $G(x_B) = x_C$, shown in Figure 4. In this case, the decomposition-through-encoding problem becomes that of resubstitution.

Let $G(x_B)$ and $F(x)$ be two functions in the MV network such that $x_B \subseteq x$. We extract block B_1 depending on x_B from F as shown in Section 3.3 but now try $G(x_B)$ as one of the decomposition functions for block B_1. If a value-reducing decomposition exists, the relation resulting from collapsing B_2 into F_2 is compared with the original representation of F. If the result is smaller, the resubstitution is accepted. If it is not smaller, the decomposition using B_1 is kept.

It may be possible to perform resubstitution directly by trying $G(x_B)$ as a Boolean divisor of F. However, the above approach shows some relations between resubstitution, decomposition, and encoding.

3.5 Common Logic Extraction

Common logic has been traditionally extracted by intersecting sets of kernels generated for logically related Boolean nodes [5]. An improvement is a specialized algorithm to manipulate double-cube and two-literal single-cube divisors [16]. However, these approaches are algebraic, limited to binary networks, and do not use flexibilities inherent in the network structure.

We propose a Boolean formulation of logic extraction based on matching common sub-functions (cofactors) in the MDD representing several logically related Boolean nodes. The matching procedure is similar to the one developed for BDD minimization [19], and is also related to the discussion of Section 4.

The idea is to find a set of functions (the merging set) that may have good common Boolean divisors. These functions are merged into a single MV node η, which replaces the merging set of nodes.

The flexibility relation for η in the new network is derived and represented as an MDD (Multi-Valued Decision

[3] We use $|v|$ to denote the number of values that signal v has.

[4] The column compatibility problem in B_1 is not considered.

Diagram)[5], with the last few variables in the order encoding the values of the function (Figure 5). After variable reordering on this MDD, we look at the cofactors depending on a fixed number (say, four) of the last fanin variables, which become the divisor variables.

Each cofactor is an MV relation. We can find a maximal set of compatible cofactors using a technique similar to the one presented in Section 4 for disjoint decomposition. The compatible cofactors are merged to form a common divisor.

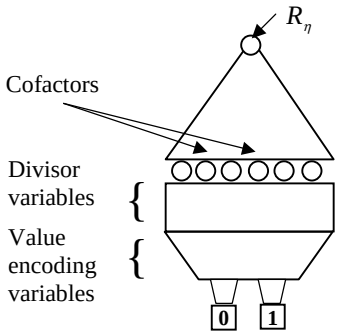

Figure 5. Extracting Boolean divisors of an MV relation.

If the divisor reduces the cost function, it is introduced into the network as a new node[6] and divided into η using Boolean division. Several compatible cofactors can be computed and tested in a sequence. Finally, η can be partially encoded.

Note that the original nodes in the merging set can be represented simply as literals of the variable representing η. This set represents one way to encode η, but we have the option of not using these and replacing them with new functions that encode η, first using partial encoding, and then possibly completing it to obtain a full binary encoding.

4 Disjoint Decomposition of an MV Relation and SAT

We reformulate the problem of finding a support-reducing disjoint decomposition of an MV relation as a SAT instance. Such decomposition with the bound set x_B exists iff the n columns in the decomposition table with variables x_B on top can be colored with no more than $n/2$ colors.

The SAT formulation uses two types of MV variables:

- n variables c_i, $1 \le i \le n$, to encode the coloring of columns. A columns may be colored with more than one color. Value j belongs to the value set of variable c_i, if the corresponding column, i, can be colored with the color j. The range of c_i is $n/2$.
- m variables d_{ij}, one for each entry in the table that can take more than one value. These variables represent subsets of original values, which are in agreement with the selected decomposition. The range of these variables is the range ρ of the relation.

The SAT clauses are now derived in a relatively straightforward way, as illustrated by the decomposition table in Figure 6. The table has entries S_{ij}, which are subsets of the range ρ. To find a support-reducing decomposition with two colors, we need four binary variables c_j to encode the coloring of the columns and, in general, 16 MV variables d_{ij} with range ρ to encode the remaining subsets of values in each entry of the table. However, some of the original subsets may be single values, so no variable is introduced for them, because the remaining subsets can only be composed of the same value.

	C_0	C_1	C_3	C_2
cd\ab	**00**	**01**	**10**	**11**
00	S_{00}	S_{01}	S_{02}	S_{03}
01	S_{10}	S_{11}	S_{12}	S_{13}
10	S_{20}	S_{21}	S_{22}	S_{23}
11	S_{30}	S_{31}	S_{32}	S_{33}

Figure 6. Support-reducing decomposition as SAT problem.

In the resulting coloring, each column has a color because in the satisfying assignment each MV variable takes at least one value. Incompatibility between two columns is enforced by the constraints depending on variables d_{ij}. For example, columns C_0 and C_1 are incompatible if at least one pair of MV variables d_{i0} and d_{i1} does not have a common value. This leads to the condition:

$$\bigvee_i (d_{i0} \cap d_{i1} = \varnothing) \Rightarrow (c_0 \neq c_1)$$

Finally, for all pairs of columns j and k, for which there exists a row i, such that $S_{ik} \cap S_{ij} = \varnothing$, we immediately have $c_j \neq c_k$. The MV expression for a pair of columns can be converted into the CNF form. The SAT instance is derived by taking the product of the constraints corresponding to all pairs of columns.

It is not surprising that we can formulate this as a SAT problem; the question is for how large a problem it is possible to perform decomposition this way.

Since SAT instances with 200 variables and 10,000 clauses are readily solved by present-day SAT solvers [14], it is reasonable to apply this approach if the nodes being decomposed are limited to about ten fanins.

There may be several cases, which can, in practice, dramatically reduce the size of the SAT instance. First, for any column, which is not compatible with any other column, we just give its own color and do not introduce additional variables. Second, if two columns can only be compatible with each other, we give these columns their own color and

[5] Actually, if we merge binary nodes and use their bits to encode the values, then the MDD is exactly the Boolean relation of these binary nodes.

[6] Note that this node is multi-valued, in general, with at most, the number of values in the merged MV node.

do not introduce variables associated with these columns. Third, the decomposition problems we encounter may only have a small percentage of entries, which have multiple values so the number of variables d_{ij} may be limited.

An extension is to find a decomposition in terms of a set of functions, like those representing gates from a gate library during technology mapping. This problem is similar to that discussed in [9] for binary functions.

5 Conclusions

This paper outlines a way of looking at several multi-level logic operations and discusses commonalities among network optimization, encoding, decomposition, resubstitution, and common logic extraction. A SAT-based formulation of one of these, decomposition, is given as an example.

The paradigm is Boolean because it uses the functional properties of nodes and is not limited to algebraic divisors generated from an SOP representation.

The distinctive features are:

- A more general theory (MV logic vs. binary logic) helps reveal some underlying relationships among the procedures.
- An efficient implementation may be possible due to a common computational cores based on a BDD package and a SAT solver.
- The Boolean operations operate on general MV relations that are derived from the network structure.

A optimization flow based on these methods could lead to improved optimization quality since:

- The use of multi-valued logic leads to searching a larger space of solutions.
- Boolean (not only algebraic) properties of nodes are exploited.
- The complete sets of don't-cares (partial cares) give greater flexibility for optimizing the nodes of the network.
- The SAT-based formulation is not limited to one particular encoding or coloring.
- Functional decomposition can be performed concurrently with technology mapping.

There is a lot of future work in this area since only the ideas of the paradigm have been outlined. We need to implement the proposed algorithms and simultaneously refine the ideas to see which heuristics work best on an extensive set of benchmarks.

Acknowledgements

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

References

[1] R. K. Brayton. Compatible Observability Don't-Cares Revisited. *Proc. ICCAD'01*, pp. 618-623.

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

[3] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proc. IEEE*, 78(2), February 1990, pp. 264-300.

[4] R. K. Brayton and S. P. Khatri. Multi-valued logic synthesis. *Proc. VLSI Design 1999*, pp. 196-206.

[5] R. Brayton and C. McMullen. The Decomposition and Factorization of Boolean Expressions. *Proc. ISCAS '82*, pp. 49-54.

[6] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comp*, C-35(8), August 1986, pp. 677-691.

[7] J. Cong and W. Long. Theory and Algorithm for SPFD-Based Global Rewiring. *Proc. of IWLS '01*, pp. 150-156.

[8] D.-J. Jongeneel, R. Otten, Y. Watanabe, R. K. Brayton. Area and Search Space Control for Technology Mapping. *Proc. DAC '00*, pp. 86-91.

[9] V. N. Kravets and K. A. Sakallah. M32: A Constructive Multilevel Logic Synthesis System. *Proc. DAC '98*, pp. 336-341.

[10] E. Lehman et al. Logic decomposition during technology mapping. *IEEE Trans. CAD*, 16(8), 1997, pp. 813-833.

[11] J. P. Marques-Silva and K. A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Trans. Comp*, 48(5), pp. 506-521, May 1999.

[12] A. Mishchenko and R. Brayton. Simplification of Non-Deterministic Multi-Valued Networks. *Proc. of IWLS*. June 2002, New Orleans.

[13] A. Mishchenko and T. Sasao. Encoding of Boolean Functions and Its Application to LUT Cascade Synthesis. *Proc. of IWLS*. June 2002, New Orleans.

[14] M. W. Moskewicz et al. Chaff: Engineering an Efficient SAT Solver. *Proc. DAC'01*, pp. 530-535.

[15] R. L. Rudell and A. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. *IEEE Trans. CAD*, 6(5), September 1987, pp. 727-750.

[16] J. Rajski, J. Vasudevamurthy, "The Test-Preserving Concurrent Decomposition and Factorization of Boolean Expressions", *IEEE Trans. CAD*, Vol.11 (6), June 1992, pp.778-793.

[17] H. Savoj, R. K. Brayton, H. Touati. Extracting Local Don't Cares for Network Optimization. *Proc. ICCAD*, 1991, pp. 514-517.

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

[19] T. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, R. K. Brayton. Heuristic Minimization of BDDs Using Don't Cares. *Proc. DAC' 94*, pp. 225-231.

[20] T. Stanion and C. Sechen. Boolean Division and Factorization Using Binary Decision Diagrams. *IEEE Trans. CAD*, 13(9), pp. 1179-1184, September 1994.