An Implicit Method for Multi-Valued Network Encoding

Jie-Hong Jiang Yunjian Jiang Robert K. Brayton
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
{jiehong,wjiang,brayton}@eecs.berkeley.edu

Abstract

Multi-valued logic optimization allows manipulating a design at a higher abstraction level. For hardware implementation, we propose an implicit method for encoding an multi-valued network into a purely binary one. It combines both input constraints and output constraints using a BDD representation. It heuristically decides which constraints to discard in case not all can be satisfied. Preliminary results show the effectiveness of the encoding algorithm. It also shows that optimization at the multi-valued logic level leads to better binary implementations.

1 Introduction

Circuit designs can be naturally expressed in terms of multi-valued (MV) logic. However circuits are rarely optimized at MV logic level before they are encoded into binary. This is due to the fact that (a) there is no good MV multi-level logic optimization package available that has a complete suite of algorithms, (b) the encoding problem is difficult for large circuits since it is hard to see how an encoding decision ultimately affects the binary logic optimizations to be applied afterwards.

Recently MV logic optimization methods have been proposed for high level implementation independent design synthesis [1, 2, 3] and a software package has been developed[4]. For hardware implementations, a final binary encoding is essential. Given a minimized network of MV logic functions, an optimal encoding transforms it into a compatible Boolean network with a minimal cost increase. The optimality can be measured in terms of the number of product-terms or literals, or other criteria such as power and testability. In this paper, the area cost is the main consideration.

Previous literature [5, 6, 7] on the encoding problem has been divided into four categories: input encoding, output encoding, input-output encoding and state encoding. Each category has a two-level and a multi-level implementation version. The theory of the input-output encoding problem for two-level implementations has been well established and efficient software packages exist [6]. Research on multi-level encoding has focused on the application to state encoding where only one MV variable appears at both the primary inputs and primary outputs; all intermediate variables are binary. To the best of our knowledge, no existing work has considered the encoding of a pure multi-level MV logic network.

In this paper, we formulate the new encoding problem in Section 2 and give an implicit encoding method in Section 3. We show through a set of experiments that MV optimization can lead to a better binary implementation if it is combined with an intelligent encoding scheme.

2 Problem Formulation

An MV combinational logic network, or *MV-network*, is a network of nodes. Each node represents a MV function with a single MV output and many MV inputs. The onset of a node f, when $f = i$, is called its "i-set". For each node, there is a default i-set which is by definition the complement of the union of the other i-sets. There is a directed edge from node i to node j, if the function at node j explicitly depends on the output variable at node i. The network has a set of primary inputs and a set of nodes which are designated as the outputs of the network. In general, any variable x_i is MV and takes on values from the set $P_i = \{0, 1, ..., |P_i| - 1\}$. Logic minimization can be performed in the MVSIS [1] environment.

We propose a circuit design scheme, where designs are specified as MV multi-level logic networks, and MV optimization is performed before it is encoded into binary. In this scheme, a larger design space is explored than the traditional approach, where an arbitrary encoding is forced at the beginning.

We argue that MV circuit optimization, followed by an encoding, followed by binary optimization is always superior to encoding followed by binary optimization. Let K be an MV circuit, and $enc(K)$ be an equivalent one using binary encoding enc. Let $Opt(enc(K))$ be an optimum multi-level implementation of K using binary encoding enc. The first observation is that in general $Opt(enc_1(K)) \neq Opt(enc_2(K))$ for two different binary encodings. Thus the best binary optimization techniques can never make up for a bad encoding. On the other hand, for any best multi-level binary implementation of a circuit, there are a set MV circuits and sets of encodings which lead to that implementation. Whether the MV circuit structures can help in choosing the encoding remains to be verified but MV optimization opens up the search space and increases the probability of this happening.

2.1 Encoding problem

Encoding Problem: *Given an optimized multi-level MV logic network with an encoding length constraint, find a binary encoding for each MV variable in the network, such that*

1. *a cost function related to the total number of cubes and literals is minimized, and*
2. *the encoding length constraint is satisfied.*

An optimal encoding needs to consider the effect of binary optimizations to be applied on the encoded network.

For intermediate MV-variables in the logic network, the encoding affects both fanin functions and fanout functions. Consider node k with fanouts $\{F_1, F_2, \ldots, F_{r_k}\}$. Each cube in the sum-of-products of the MV functions at $F_i, i \in \{1, \ldots, r_k\}$ forms a "face constraint" for k: let k have a range of P_k values; a face constraint is a subset of the values for k that are to be assigned to one face of a binary P_k-dimensional cube, without any other value sharing the same face. The function at node k itself, the i-sets, gives dominance constraints for logic minimization. Additional constraints are given according to the size of each i-set. For notations and definitions of general encoding problems refer to [5]. Techniques for traditional input-output encoding and state encoding [6, 8, 9, 10] do not consider input and output constraints from different nodes.

3 Implicit Encoding

We propose an implicit method to combine both input and output constraints in a BDD representation. Then the satisfiability of these constraints can be easily tested. A solution is obtained by finding a path from the root node to the constant one terminal node in the corresponding BDD.

We visit each MV-node in the network in a reverse topological order from primary outputs to primary inputs. At each MV-node, we use Procedure 1 to compute a binary encoding.

Procedure 1 *(Implicit encoding for MV-node i)*

1. *For each fanout node k, generate input face constraints from node k;*
2. *Generate output constraints from node i;*
3. *Assign a weight for each constraint according to an estimated cost and sort all constraints based on their weights;*
4. *Construct the constraint BDD incrementally in the sorted order until no more constraints can be satisfied;*
5. *Derive an encoding solution from the BDD representation;*
6. *Generate binary nodes from the code assignment and update the network.*

The encoding length constraints can be given by the user. As a default value, we use the minimum binary code length; for an N-valued variable, $\lceil log_2 N \rceil$ is used as the default code length. To encode an N-valued variable with with m bits, we create $N * m$ Boolean variables in the BDD representation. Three sets of constraint formulas are generated for the encoding problem: uniqueness constraints, input constraints and output constraints.

3.1 Uniqueness constraints

These are used to ensure that each value of the MV-variable is encoded with a unique binary code; for each pair of the values, $i, j \in N$, at least one of their m bits should be different. We generate the following formulae:

$$\bigwedge_{i,j \in N, i \neq j} \bigvee_{k=0}^{m-1} (b_{ki} \oplus b_{kj})$$

where b_{ki} represents the k^{th} bit of the binary code for value i.

3.2 Input constraints

The formula for input constraints is a conjunction of the formula for each input face constraint. We generate a face constraint matrix from the sum-of-products representation of the fanout nodes. A face constraint is generated for each cube in the fanout function, which depends on a subset of its values. For example, suppose variable x is a fanin of f, where

$$f = a\{0,1\}x\{0,2\} + a\{2\}x\{0,1,3\}$$

$x\{0,2\}$ is a MV literal, which evaluates to 1 if x takes either value 0 or 2; cube $a\{0,1\}x\{0,2\}$ is the conjunction of the two literals and evaluates to 1 if both are 1. Two face constraints are generated from f, which form the constraint matrix C below, where each face forms a row and a 1 represents the inclusion of the corresponding value in the face.

$$\begin{bmatrix} 0,2 \\ 0,1,3 \end{bmatrix} \Rightarrow C = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 2 & 3 & 4 \end{bmatrix}$$

The Boolean formula for each face constraint (row in the matrix) is a conjunction of sub-formulae that distinguish the code for values with 1 in the row from the code for values with 0 in the row. Let m be the number of binary bits used for encoding and M be the set $\{0, 1, \ldots, m-1\}$. For a given row, let N_0 be the set of columns that has a 0, and N_1 be the set of columns with a 1. The encoding is given by bits $\{b_{i,j}\}$ where $(b_{0,j}, b_{1,j}, \ldots b_{m-1,j})$ is the code for value j.

$$
\begin{aligned}
F_z(k,l) &= (\wedge_{i,j \in N_1} b_{ki} \bar{\oplus} b_{kj}) \wedge (\wedge_{s \in N_1} b_{ks} \oplus b_{kl}) \\
F_{face} &= \bigwedge_{l \in N_0} \bigvee_{k \in M} F_z(k,l)
\end{aligned}
$$

$F_z(k,l)$ says the k^{th} binary bit for all 1 columns must be the same and this must be different from the l^{th} 0 column. For the example given earlier, where $M = \{0,1,2\}$, $N_0 = \{1,3,4\}$ and $N_1 = \{0,2\}$, the following constraint formulae are created for the first face [10100]:

$$
\begin{aligned}
F_z(M,1) &= [(b_{00} \bar{\oplus} b_{02}) \wedge (b_{00} \oplus b_{01})] \vee \\
&\quad [(b_{10} \bar{\oplus} b_{12}) \wedge (b_{10} \oplus b_{11})] \vee \\
&\quad [(b_{20} \bar{\oplus} b_{22}) \wedge (b_{20} \oplus b_{21})] \\
F_z(M,3) &= [(b_{00} \bar{\oplus} b_{02}) \wedge (b_{00} \oplus b_{03})] \vee \\
&\quad [(b_{10} \bar{\oplus} b_{12}) \wedge (b_{10} \oplus b_{13})] \vee \\
&\quad [(b_{20} \bar{\oplus} b_{22}) \wedge (b_{20} \oplus b_{23})] \\
F_z(M,4) &= [(b_{00} \bar{\oplus} b_{02}) \wedge (b_{00} \oplus b_{04})] \vee \\
&\quad [(b_{10} \bar{\oplus} b_{12}) \wedge (b_{10} \oplus b_{14})] \vee \\
&\quad [(b_{20} \bar{\oplus} b_{22}) \wedge (b_{20} \oplus b_{24})] \\
F_{face} &= F_z(M,1) \wedge F_z(M,3) \wedge F_z(M,4)
\end{aligned}
$$

3.3 Output constraints

We first consider dominance constraints. Dominance relations between values are used to simplify their expressions. Suppose after encoding, the code for value i dominates that of value j, denoted as $i > j$. An example is $i = 0111$ and $j = 0101$. Then the onset of the function for value i (called the *i-set* of the function) can be treated as don't cares to minimize the function for value j. Relation $i > j$ can be expressed as $\wedge_{k=0}^{m-1}(b_{ki} \vee \overline{b_{kj}})$. As an example, assume variable z takes on values $\{0,1,2,3\}$. Suppose a dominance relation, $0 > 2, 0 > 1, 1 > 3$, gives the most savings in cubes and literals. We generate the following Boolean formula:

$$
\begin{aligned}
& [(b_{00} \vee \overline{b_{02}}) \wedge (b_{10} \vee \overline{b_{12}})] \\
\wedge \; & [(b_{00} \vee \overline{b_{01}}) \wedge (b_{10} \vee \overline{b_{11}})] \\
\wedge \; & [(b_{01} \vee \overline{b_{03}}) \wedge (b_{11} \vee \overline{b_{13}})]
\end{aligned}
$$

In addition to dominance constraints, we use a heuristic to restrict cube duplication. After encoding, m binary nodes are created to replace the MV node. The on-set functions of the k^{th} binary node is the union of various i-sets whose binary code has a 1 in the k^{th} bit. For example, suppose value j is given code 0101; then its i-set is part of bits 1 and 3. In other words, most i-sets are duplicated by several binary node. Our goal is to minimize the duplication of the larger i-sets.

For each MV variable, we order its values in descending order according to the number of cubes in the corresponding i-set. Then the heuristic assigns fewer 1's to higher order i-sets, which will cause less binary nodes to include this i-set. The default i-set, the one not represented by the MV-network representation, is normally large and should be avoided; thus we insert the default i-set as the first element in the ordered list and it will be encoded as the off-set for all binary bits.

In practice, the heuristic for restricting cube duplication is more relevant to the final binary implementation than the dominance constraints. In the experiments, we use the heuristic to constrain the output code assignment and generate don't cares from the dominance relation that results.

3.4 Constraint selection

If a solution exists, we find an optimal one by selecting a shortest path from the root to the constant one terminal node in the corresponding BDD. By selecting the shortest path, more bit assignments are don't cares, and these can be explored to optimize the generated binary nodes. If the constraints cannot be satisfied simultaneously, we select a satisfiable subset such that the increased overhead is minimized. We use the following simple heuristic to select more preferable constraints.

We consider two factors: (i) the number of times that an input constraint is shared between multiple functions; (ii) the number of 1's in an input constraint. The more frequent a constraint is shared, the higher priority should be given to satisfy it. If not satisfied, a face constraint with more 1's would possibly break up into more cubes, thus should be given a higher priority.

The constraint formulas are added into the BDD representation one at a time from highest to lowest priority, until no additional constraint can be satisfied. Thus the BDD is incrementally built only once.

4 Experiments

We implemented the implicit encoding method in MVSIS [1]. The examples we experimented with, come mostly from the MV logic benchmark suite available from Portland State University [11]. The target of our experiments was to (a) measure the relative merits of MV optimization before binary implementation, and (b) measure the effec-

Table 1: Encoding result comparison

	mvno-impl		mvno-nat		mvopt-impl		mvopt-nat	
examples	cubes	lits	cubes	lits	cubes	lits	cubes	lits
TT-cell	40	88	40	88	40	88	40	88
adder_mod4	16	32	16	28	16	32	16	28
alg5	46	85	51	114	17	42	27	71
aluack	21	44	22	50	22	52	22	45
car	25	50	31	71	23	42	26	55
com35-45	21	43	25	45	15	31	16	36
conv45	17	36	15	34	15	37	14	35
employ1	17	42	21	51	19	48	24	62
employ2	19	54	58	131	31	82	64	144
ex2	23	51	23	61	23	51	23	61
ex3	24	52	24	62	24	52	24	62
iris	49	180	50	164	49	180	50	164
matmul	74	168	100	228	82	193	100	228
max	113	236	149	292	107	176	91	159
mm3	13	28	22	60	13	28	22	60
mm4	27	50	26	47	27	50	26	47
mm5	152	353	116	207	109	232	113	236
mmp5	74	164	76	172	71	198	73	210
monks2tr	50	144	58	173	49	149	58	173
monks3tr	42	135	42	135	44	129	42	135
monksltr	47	145	49	150	47	145	49	150
nursery	44	96	51	115	30	88	53	113
pal3x	78	184	61	144	47	130	48	135
plus5	25	65	23	60	25	65	23	60
rot-alu	29	60	30	63	34	85	33	65
xor-all-mux	18	39	15	35	19	37	19	42
xor-rotall	12	24	12	22	12	24	12	22
normalized	1	1	1.07	1.10	0.94	0.97	1.01	1.06

tiveness of our implicit encoding scheme relative to the natural encoding ($0-000$, $1-001$, $2-010$, $3-011, \ldots,$).

For each benchmark example, we encode the entire MV network including the primary inputs and primary outputs. We generate two networks, one with an MV optimization script applied and the other where no MV optimization is done. These are then encoded in two ways: by our implicit encoding method and a natural encoding where the natural number code is used. All four sets of results are then optimized using the binary optimization script, `script.rugged`, in SIS[12] and we record the number of cubes and literals that result. The four groups of results are shown in Table 1.

The following script is used for MV optimization:

```
sweep; eliminate 0; simplify
eliminate 20; simplify; reset_default
fullsimp; pair_decode 10
eliminate 0; simplify
pair_decode 10; eliminate 0
simplify; decomp
simplify; eliminate 0
fullsimp; fx -g
eliminate 0; fullsimp
```

After encoding of an MV-node, don't cares are generated from the output dominance constraints to simplify the binary nodes.

In Table 1 Columns `cubes` and `lits` show the total number of cubes in the sum-of-products forms and literals in the factored forms, in the final results minimized by SIS. The results are grouped under `mvno-impl`, `mvno-nat`, `mvopt-impl` and `mvopt-nat`, where `mvno` represents no MV optimizations, `impl` represents our implicit encoding, and `nat` represents the natural encoding.

From the results in Table 1, we make the following observations:

- The implicit method outperforms the natural encoding in most of the examples. On average it leads to about 10% saving in both cube count and literal count in the final optimized binary network.
- MV optimization leads to a better binary implementation, if we compare `mvopt-impl` with `mvno-impl`. This works extremely well for the arithmetic examples (`alg5, max, mm5`). For smaller examples, where a two-level function is the best implementation at the MV level, this does not make much difference. On average, using the implicit encoding, the final cube count is 6% better and literal count is 3% better.
- Comparing the results between `mvopt-impl` and `mvno-nat` (which would be the classical method used most), we see that MV optimization combined with the implicit encoding is about 12% better in the final optimized binary network. This shows the effectiveness of our proposed synthesis scheme.

We also compare the two encoding methods without any MV or binary optimization. In normalized results for the same set of examples, the implicit method is better than the natural encoding by 34% in cube count and 45% in literal count (in the MV network). This combined with Table 1 shows that (i) the implicit method is much better if the immediate saving is considered, (ii) the immediate saving may be obscured by binary optimizations.

One of the major advantages of the implicit method is that constraints can be easily added or removed from consideration. The satisfiability of the constraints is implicit in the BDD structure. As a result, BDD size affects the efficiency of the implicit encoding. To reduce the BDD size by good variable ordering, we group symmetric variables together, i.e. variables representing the code for the same symbol are grouped together. However, when the number of BDD variables is larger than about 30, the BDD operations proceed slowly. Therefore, to limit the number of BDD variables, we use the minimum length encoding for all MV-variables; when an MV-variable has more than a given number of values, we switch to the natural encoding.

5 Conclusions and Future Work

We analyzed the encoding problem for a multi-level MV logic networks, and proposed an implicit method for solving it. Our experiments show that our encoding method outperforms the natural encoding, and more importantly, MV optimization combined with an intelligent encoding can lead to a better final implementation than use of the natural encoding at the beginning.

Future research includes refining the encoding algorithm and applying it to other applications. We would like to explore the following. (i) In constraint selection, a large constraint may be partitioned into sub-constraints that are easier to satisfy. (ii) In constraint BDD construction, heuristics need to be experimented with in order to compare the relative merits between input and output constraints. (iii) For MV-variables with a large number of values, a hybrid approach can be applied, where the first few binary bits are heuristically decided and the rest bits are assigned using the implicit method when the complexity is manageable. (iv) The encoding scheme can be applied to the state encoding problem with multi-level implementations.

Acknowledgement

We are grateful for the support of the SRC under contract 683.004 and the California Micro program and industrial sponsors, Fujitsu, Cadence, Motorola and Synopsys.

References

[1] M. Gao, J. Jiang, Y. Jiang, Y. Li, S. Singha, and R. K. Brayton, "MVSIS," in *Proc. of the Intl. Workshop on Logic Synthesis*, May. 2001.

[2] 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.

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

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

[5] T. Villa, T. Kam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, *Synthesis of Finite State Machines: Logic Optimization*. Kluwer Academic Publishers, 1997.

[6] T. Villa and A. L. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 905–924, Sept. 1990.

[7] G. D. Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Trans. Computer-Aided Design*, vol. 4, pp. 269–85, July. 1985.

[8] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "A Fully Implicit Algorithm for Exact State Minimization," in *Proc. of the Design Automation Conf.*, pp. 684–690, June 1994.

[9] E. Goldberg, T. Villa, , R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Theory and algorithms for face hypercube embedding," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 472–88, June 1998.

[10] A. Saldanha, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Satisfaction of input and output encoding constraints," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 589–602, May 1994.

[11] Portland Logic Optimization group, "Portland state university." `http://www.ee.pdx.edu/p̃olo/`.

[12] 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 Laboratory, Univ. of California, Berkeley, CA 94720, May 1992.