Center for
Reliable
Computing

TECHNICAL REPORT

An Output Encoding Problem And A Solution Technique

Subhasish Mitra, LaNae J. Avra and Edward J. McCluskey

97-1 (CSL TR # 98-761) November, 1997	**Center for Reliable Computing** Gates Building 2A, Room 236 Computer Systems Laboratory Dept. of Electrical Engineering and Computer Science Stanford University Stanford, California 94305-9020

Abstract:

We present a new output encoding problem as follows: Given a specification table, such as a truth table or a finite state machine state table, where some of the outputs are specified in terms of 1s, 0s and *don't cares*, and others are specified symbolically, determine a binary code for each symbol of the symbolically specified output column such that the total number of output functions to be implemented after encoding the symbolic outputs and compacting the output columns is minimum. There are several applications of this output encoding problem, one of which is to reduce the area overhead while implementing scan or pseudo-random BIST in a circuit with one-hot signals. This algorithm can also be used as a pre-processing step during FSM state encoding. In this report, we develop an exact algorithm to solve the above problem, prove its correctness, analyze the worst case time complexity of the algorithm and present experimental data to validate the claim that our encoding strategy helps to reduce the area of a synthesized circuit. In addition, we have investigated the possibility of using elementary gates to facilitate further merging of the output functions generated by the encoding bits with the output functions generated by the elementary gates.

Funding:

This work was supported by the Advanced Research Projects Agency under prime contract No. DABT63-94-C-0045.

Imprimatur: Nirmal Saxena and Jonathan T. Y. Chang

An Output Encoding Problem And A Solution Technique

Subhasish Mitra, LaNae J. Avra and Edward J. McCluskey

CRC Technical Report No. 97-1
(CSL TR No. 98-761)
November, 1997

Center for Reliable Computing
Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University, Stanford, California 94305

Abstract

We present a new output encoding problem as follows: Given a specification table, such as a truth table or a finite state machine state table, where some of the outputs are specified in terms of 1s, 0s and *don't cares*, and others are specified symbolically, determine a binary code for each symbol of the symbolically specified output column such that the total number of output functions to be implemented after encoding the symbolic outputs and compacting the output columns is minimum. There are several applications of this output encoding problem, one of which is to reduce the area overhead while implementing scan or pseudo-random BIST in a circuit with one-hot signals. This algorithm can also be used as a pre-processing step during FSM state encoding. In this paper, we develop an exact algorithm to solve the above problem, prove its correctness, analyze the worst case time complexity of the algorithm and present experimental data to validate the claim that our encoding strategy helps to reduce the area of a synthesized circuit. In addition, we have investigated the possibility of using elementary gates to facilitate further merging of the output functions generated by the encoding bits with the output functions generated by the elementary gates.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Column compaction plays a major role in the synthesis of digital systems [Wei 87]. In *column compaction*, the number of output functions for a given specification is reduced by merging the outputs which are logically equivalent, or can be made equivalent through assignment of *don't cares*. Output column i is *logically equivalent* to output column j if and only if for all inputs, if i is 1 (0) then j is also 1 (0). Given a specification table (a truth table for a combinational function or a state table for a sequential function), logically equivalent outputs can be merged (compacted). Output columns i and j are said to be *compatible* if and only if for all inputs either they are equal or at least one of them has a *don't care* entry. Two compatible outputs can be merged (compacted) by appropriately fixing the *don't care* entries to 0 or 1 so that they become logically equivalent. Two examples of output column compaction are shown in Fig 1. Not only logically equivalent functions can be compacted; logically complementary functions can also be compacted. Output column i is *logically complementary* to output column j if and only if for all inputs, if i is 1 (0) then j is 0 (1). When two logically complementary outputs are merged (compacted), in the final implementation, we derive one of them from other by using an inverter. Given a set of output columns, the problem of finding the smallest set that can be obtained by compacting the given set can be related to the *Maximum Clique Partitioning Problem*, which is an NP-Complete problem [Wei 87]. This smallest set of compacted outputs is called the *minimum cardinality output column cover*. Column compaction can greatly reduce circuit area; this is true both for the PLA implementation of the circuit [Brayton 84] and for multi-level implementation of logic circuits.

$$\begin{array}{cc}0&0\\1&1\\1&1\\0&0\end{array} \longrightarrow \begin{array}{c}0\\1\\1\\0\end{array} \qquad \begin{array}{cc}0&-\\-&1\\1&1\\0&-\end{array} \longrightarrow \begin{array}{c}0\\1\\1\\0\end{array}$$

Figure 1. Examples of column compaction

The different types of encoding problems include the *finite state machine (FSM) state encoding* problem and the *input* and the *output* encoding problem. McCluskey and Unger pointed out the symmetries among the different encodings of the states of a finite state machine and developed a formula for the number of *distinct* encodings [McCluskey 59]. Techniques for FSM state encoding have been discussed in [Ashar 91] [Dolotta 64] [De Micheli 84] [De Micheli 85] [Du 91] [Lin 90] [Villa 90] [Tumbush 74]. The input encoding problem has been reported in [Buijs 91] and [Yang 91]. One version of the output encoding problem, with a goal to reduce the number of product terms of a logic function, has been discussed in [Devadas 91]. Another version of the output encoding problem is discussed in [Saldanha 88]. A heuristic solution to the output encoding problem which uses column compaction to reduce the number of outputs after the symbols are encoded is given in [Binger 91]. However, this algorithm does not consider any previously-encoded binary outputs when determining the encoding for the symbols of the symbolic output column.

In this report, we propose an output encoding problem whose objective is different from those discussed in the literature. The input to our problem is a specification table of a combinational (or sequential) circuit in terms of a truth table (or state table) such that some of the outputs are specified in terms of 0s, 1s and *don't cares* while the other outputs are symbolically specified. The problem is to encode the symbols in the symbolic output column, so that after encoding, the *maximum* number of newly generated output functions become logically equivalent to, or can be made logically equivalent to (taking the advantage of the *don't cares*), the pre-existent output functions that had already been specified in terms of 1s, 0s and don't cares; in that case, the cardinality of the output column cover computed using the principle of column compaction will be minimum. An outline of the algorithm has been presented in [Mitra 97b]. In fact, our algorithm can be used for merging the output functions generated out of an encoding with the inputs of the given specification. There are various applications of this output encoding problem. Section 2 explains the motivation behind studying this type of an output encoding problem. Section 3 presents our output encoding algorithm for the case in which the specification table does not contain any *don't cares* in its output part. In Sec. 4, we extend the algorithm given in Sec. 3 to handle *don't cares* in the outputs of the specification. Experimental results are reported in Sec. 5. In Sec. 6, we extend our algorithm, presented in Sec. 3 and 4, to handle elementary gates and achieve further reduction of the number of output functions. Finally, we conclude in Sec. 7.

2. MOTIVATION

In this section, we explain the motivation behind studying our output encoding problem. Consider the specification shown in Table 1. Output columns c_1, c_2, c_3 and c_4 of Table 1 are specified in terms of 1s, 0s and don't cares. They constitute the *B-set*, the bound set. The last output column of Table 1 is symbolic — it is referred to as the *S-Column*, the symbolic column. For this report, we will consider specification tables containing a single symbolic output column (S-Column) for simplicity. Multiple symbolic output columns can be handled by choosing an appropriate ordering of the symbolic outputs and repeatedly applying the algorithm reported in this report.

For Table 1, since we have seven distinct symbols in the S-column, we need at least 3 bits to encode them. The B-set = {c_1, c_2, c_3, c_4}. Table 2(a) shows one possible encoding of the symbols in the S-column and Table 2(b) shows the truth table for the corresponding function to be realized. Note that in Table 2(b), c_5, c_6 and c_7 are generated due to the encoding of the symbols. Even if column compaction is performed on Table 2(b), the number of output columns cannot be reduced. This is because none of c_5, c_6 or c_7 can be made equivalent to the members of the B-set.

Table 1. Partial truth table with symbolic output

Input	Bound Outputs c_1 c_2 c_3 c_4	Symbolic output
10101	1 0 1 0	X_1
01100	0 0 1 0	X_2
10001	1 0 0 1	X_3
01111	- 1 - 0	X_2
11110	0 - 0 -	X_4
01010	1 1 - 0	X_5
11111	1 0 - 0	X_1
1110-	0 - 1 1	X_6
11011	1 0 1 1	X_7
01001	0 1 0 1	X_4

Table 2(a). Symbol Encoding (Table 1)

Signals	Encoding
X_1	100
X_2	111
X_3	101
X_4	011
X_5	010
X_6	110
X_7	001

Table 2(b). Truth Table obtained after the encoding

Input	Outputs c_1 c_2 c_3 c_4 $\mathbf{c_5}$ $\mathbf{c_6}$ $\mathbf{c_7}$
10101	1 0 1 0 1 0 0
01100	0 0 1 0 1 1 1
10001	1 0 0 1 1 0 1
01111	- 1 - 0 1 1 1
11110	0 - 0 - 0 1 1
01010	1 1 - 0 0 1 0
11111	1 0 - 0 1 0 0
1110-	0 - 1 1 1 1 0
11011	1 0 1 1 0 0 1
01001	0 1 0 1 0 1 1

It is possible to reduce the number of output columns after column compaction if we encoded the symbols of Table 1 in a different way. This is shown in Table 3(a). The corresponding truth table after column compaction is shown in Table 3(b). We find that in Table 3(b), all the extra output columns generated due to the encoding of the symbols of Table 1 have been merged with the already existent output columns and thus the encoding of Table 3(b) gives the minimum cardinality of the output column cover. Note that, in Table 3(b), columns c_1, c_3 and c_4 represent the symbolic outputs.

Our aim is to encode the symbols in the S-column so that the output columns generated by the encoding can be maximally merged with the output columns in the B-set by column compaction. By maximal merging we mean that the cardinality of the output column cover obtained after encoding the symbols is minimum.

Table 3(a). Another symbol encoding

Signals	Encoding
X_1	110
X_2	010
X_3	101
X_4	001
X_5	100
X_6	011
X_7	111

Table 3(b). Corresponding Truth Table

Input	Output $\mathbf{c_1}$ c_2 $\mathbf{c_3}$ $\mathbf{c_4}$
10101	1 0 1 0
01100	0 0 1 0
10001	1 0 0 1
01111	0 1 1 0
11110	0 - 0 1
01010	1 1 0 0
11111	1 0 1 0
1110-	0 - 1 1
11011	1 0 1 1
01001	0 1 0 1

This output encoding problem has many applications. Digital systems are often specified in terms of a combination of binary-encoded and symbolic signals. Let us consider a finite state machine, a subset of whose output signals are control signals for a selector implemented using transmission gates or a set of tristate buffers connected to a bus and hence, should be one-hot encoded. While testing the circuit using random patterns in a scan or pseudo-random BIST environment [McCluskey 86], *don't care* states may appear in the FSM bistables, causing the output signals to violate the one-hot requirement. We can avoid this situation by designing the FSM such that it generates a set of fully encoded output signals instead of the one-hot output signals and consequently pass these encoded signals through a decoder to generate the one-hot signals. This will ensure that the one-hot signals remain one-hot even if the bistables reach an invalid state during testing. This scheme is illustrated in Fig 2. Table 4(a) shows the output part of the state table of such an FSM. Outputs c_5, c_6, c_7 and c_8 are one-hot output signals. We specify the four one-hot signals symbolically, and the corresponding specification table is shown in Table 4(b). Next, we can apply our output encoding algorithm to minimize the number of output columns in the resulting FSM. Depending on the encoding performed, we specify the truth table of the decoder to decode these encoded signals and generate the four one-hot signals. This scheme has been described in details in [Mitra 97a]. However, our algorithm is not restricted to this application only — it is applicable to any specification table that has some output columns specified using 1s, 0s and don't cares and other output columns with symbolically specified entries. The algorithm is useful for encoding the mnemonic output fields of the micro-codes in order to reduce the width of the micro-control memories. This algorithm can also be used as a pre-processing step during FSM state encoding.

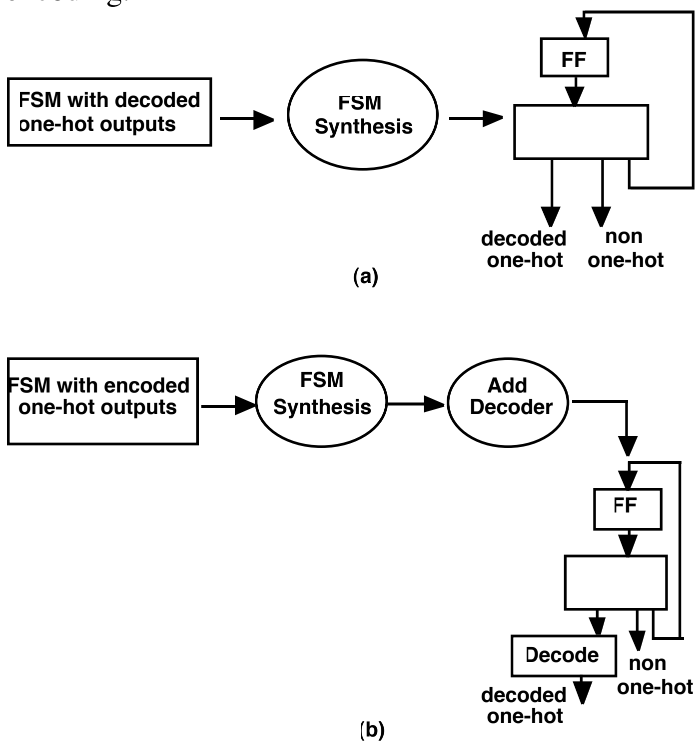

Figure 2(a). Conventional FSM synthesis scheme Figure 2(b). FSM synthesis scheme to ensure one-hot condition on control signals during scan or BIST

Table 4(a). Output part of FSM generating one-hots

FSM Outputs	
c_1 c_2 c_3 c_4	c_5 c_6 c_7 c_8
- 0 1 -	1 0 0 0
- 0 1 0	0 1 0 0
0 1 - 1	0 0 0 1
1 - - 1	0 1 0 0
1 - - 1	0 0 1 0
0 0 1 0	1 0 0 0
- 1 0 -	0 0 1 0
1 1 - -	0 0 0 1

Table 4(b). Symbolic one-hot outputs of Table 4(a)

Outputs c_1 c_2 c_3 c_4	Symbolic Output
- 0 1 -	X_1
- 0 1 0	X_2
0 1 - 1	X_4
1 - - 1	X_2
1 - - 1	X_3
0 0 1 0	X_1
- 1 0 -	X_3
1 1 - -	X_4

3. THE ENCODING ALGORITHM FOR FULLY SPECIFIED OUTPUTS

In this section, for simplicity, we present our output encoding algorithm with the assumption that the output columns belonging to the B-set are fully specified with 1s and

0s. In Sec. 4, we will extend the algorithm to consider the case where the elements of the B-set may contain *don't cares*. A set of theorems, described below, forms the basis of our algorithm.

The first step of the algorithm filters out a few of the elements of the B-set by making a consistency check. The consistency criterion is given by the Theorem 1.

Theorem 1 : Consistency Check

If there exist two rows, p and q, in the given specification table for which the S-column has the same value X_j and $c_i \in$ B-set has different values, then c_i cannot contribute to reducing the number of output columns after encoding the S-column. In this case, c_i is said to be *inconsistent* with respect to X_j.

Proof : Without loss of generality, let us assume that c_i has a '1' in the row p and a '0' in the row q and the S-column contains X_j in both of these rows. For any encoding of the S-column, all columns generated by this encoding will have equal values (both '0' or both '1') in the entries corresponding to row p and row q (since S-column has the same value in row p and row q of the specified table). Thus, column c_i can never match any column generated by any encoding of the S-column. Hence, c_i cannot help in reducing the output column cover after encoding the S-column. Q.E.D.

We illustrate the application of Theorem 1 using the example in Table 5(a). As specified in Table 5(a), rows 2 and 4 have the same value of the symbolic output (X_2) (S-column). However, rows 2 and 4 have '0' and '1', respectively, in column c_1. For any encoding of X_2 , say 01, the newly-generated columns have the same value in the rows 2 and 4 and hence can never be merged with column c_1 (Table 5(b)).

Table 5(a). Output part of a specification table

c_1	c_2	c_3	Symbolic
1	0	1	X_1
0	0	1	X_2
1	1	0	X_3
1	0	1	X_2
0	1	0	X_4

Table 5(b). Table of 5(a) with encoded symbols

c_1	c_2	c_3	Encoded
1	0	1	00
0	0	1	01
1	1	0	10
1	0	1	01
0	1	0	11

We apply Theorem 1 to each c_i in an effort to reduce the subset of the elements of the B-set considered when encoding the symbols in the S-column. The reduced subset, called the *reduced B-set*, contains no inconsistent columns.

Now, we consider the output portion of the specified table consisting of the columns which are the elements of reduced B-set and the S-Column. Let us call this table the *consistent output table* (COT). An example of such a table is shown in Table 6(a). We can perform row merging on this consistent output table to obtain a *reduced consistent output table* (RCOT) using Theorem 2.

Theorem 2 : Row Merging

In the consistent output table, all rows having the same values for the S-Column are equivalent and hence, can be merged.

Proof : The proof is straightforward. Consider any two rows p and q of COT having the same value in the S-Column. Since any column (except the S-column) of COT is a member of reduced B-set, by Theorem 1, any other column of COT will also have the same value in the entries corresponding to rows p and q. Hence, in the COT, rows p and q are equivalent and can be merged without losing any information. Q.E.D.

By applying Theorem 2 to the COT for each value of the symbol in the S-column, we obtain the RCOT from the COT. Let the number of rows in RCOT be n. This means that there are n distinct symbols in S-Column which can be encoded using $m = \lceil \log_2(n) \rceil$ bits. Next, we remove the S-Column from RCOT to obtain the *Reduced Consistent-Bound Output Table* (RCBOT) as shown in Table 6(b). Once we obtain the RCBOT, we attempt to reduce the number of columns in the RCBOT using Theorem 3.

Table 6(a). The Consistent Output-Table

c_2	c_3	Symbolic
0	1	X_1
0	1	X_2
1	0	X_3
0	1	X_2
1	0	X_4

Table 6(b). The Reduced Consistent Bound Output Table

c_2	c_3
0	1
0	1
1	0
1	0

Theorem 3 : Single Column Counting Check

Any column of RCBOT having more than 2^{m-1} 1s (0s) cannot contribute to reducing the size of the output column cover after encoding the symbols, where m is the number of bits used to encode the symbols in the S-column. For encoding using the minimum number of bits, $m = \lceil \log_2 n \rceil$, where n is the number of symbols to be encoded (equal to the number of rows in the RCBOT).

Proof : The proof is straightforward. For any encoding of the n symbols using m bits, m columns $e_1, e_2, ..., e_m$ corresponding to the encoding bits are generated in the RCOT. It is obvious that the number of 1s (or 0s) in each e_i column ($1 \leq i \leq m$) is less than or equal to 2^{m-1} because otherwise we will end up having the same binary code assigned to two distinct symbolic outputs. Hence, any column of RCOT which is a member of reduced B-Set (and hence a column of RCBOT) having the number of 1s (or 0s) exceeding 2^{m-1} cannot be merged with any of the e_i ($1 \leq i \leq m$) and hence cannot help in reducing the size of the output column cover. Q.E.D.

Theorem 3 is illustrated in Table 7. The RCBOT corresponding to Table 7 has four distinct rows. If we use the minimum number of bits to encode the symbolic outputs, then $m = 2$. Columns c_1 and c_2 do not satisfy the upper bound on the number of 1s and 0s, respectively. Hence, they cannot help in reducing cardinality of the output column cover after the encoding of the four symbols has been performed. We call the set of columns satisfying Theorem 3, the *Column-Count-Set-1* (CCS-1). The CCS-1 for the example in Table 7 is $\{\{c_3\}, \{c_4\}\}$. Using the CCS-1, we construct a family of sets CCS-i using Theorem 4, which is a generalized version of Theorem 3. In general, each member of CCS-i is a set of i columns which satisfies *i-Column-Counting-Check* (Theorem 4).

Table 7. A **RCBOT** to illustrate Theorem 3

c_1	c_2	c_3	c_4
1	1	0	1
0	0	0	1
1	0	1	0
1	0	1	0

Theorem 4 : *i* - Column Counting Check

Any set of i columns of RCBOT having any of the 2^i possible binary combinations appearing more than 2^{m-i} times in the rows of the RCBOT cannot help to reduce the size of the output column cover by i after encoding the symbols. Here m is the number of bits used to encode the symbols. For encoding using the minimum number of bits, m =

$\lceil \log_2 n \rceil$, where n is the number of symbols in the S-column, to be encoded (number of rows of **RCBOT**).

Proof : The proof is straightforward and follows from the proof of Theorem 3. The key point of the proof is that in any set of n distinct binary strings of length m, any binary string of length i ($\leq m$) can occur atmost 2^{m-i} times under a specified set of i columns.

All the sets of i columns that satisfy the i-Column Counting check form elements of CCS-i. It can be shown, that for any set of columns of RCBOT which is an element of CCS-$(i+1)$, all its subsets of cardinality i are members of CCS-i ($i \geq 1$). Hence, a candidate for CCS-$(i+1)$ is generated by taking the union of an element of CCS-i and an element of CCS-1 such that the resulting set has cardinality equal to $(i + 1)$. We apply Theorem 4 iteratively for increasing i until either CCS-i is empty or i is equal to the number of encoding bits. In either case, we choose a member of CCS-j, j being the largest integer such that CCS-j is not empty, and encode the first j bits of the symbol under the S-column in row k of the RCOT with the binary values from row k of the RCOT that correspond to the binary-encoded columns that are members of CCS-j. The encodings of the first j bits are not necessarily unique over all the symbols. We encode the remaining (m-j) bits of the symbols in such a way that the m bit encoding of each symbol is unique.

Figure 3 illustrates the high-level flow of our algorithm (Algorithm 1), showing how Theorems 1 to 4 are applied to a specification table. First, Theorem 1 is applied to the input specification table to obtain the Consistent Output Table (COT). Next, Theorem 2 is applied to the COT to merge equivalent rows and generate the Reduced Consistent Output table (RCOT) and the Reduced Consistent Bound Output Table (RCBOT). Theorem 3 is applied on RCBOT to obtain CCS-1. Next, additional CCS-i's are calculated from the RCBOT using Theorem 4.

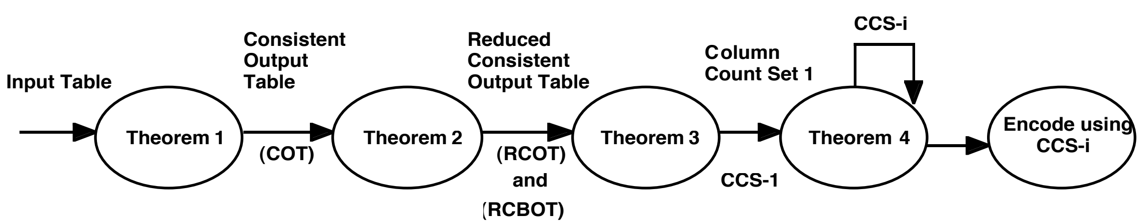

Figure 3. A High Level Flow of our output encoding algorithm (Algorithm 1)

ALGORITHM 1: Encode Symbols

Input : The output portion of the given specification table; one column is for symbolic output and the other columns are fully specified binary outputs.

Output : An encoding of the symbols in the symbolic output column such that the cardinality of the output column cover computed after encoding the symbols is minimum.

Steps :

1. Apply Theorem 1
for each column $c_k \in$ **B-set**
Check if c_i is an *inconsistent column*
2. Form the **Consistent Output Table** (COT) consisting of the *consistent* columns and the **S-Column**.
3. Apply Theorem 2
Merge equal rows of the **COT** and obtain **Reduced Consistent Output Table** (RCOT) and the **Reduced Consistent Bound Output Table** (RCBOT)
4. Let $m = \lceil \log_2 n \rceil$, where n is the number of rows of RCOT.
5. Apply Theorem 3
CCS-1 = *NULL*;
for each column c_i of RCBOT
if ((0s count in column $c_i \leq 2^m / 2$) AND (1s count in column $c_i \leq 2^m / 2$))
CCS-1 = CCS-1 $\cup \{c_i\}$
6. Apply Theorem 4
$j = 1$
while (($j < m$) AND (CCS-j is not *NULL*))
CCS-$(j+1)$ = *NULL*;
for each $X_k \in$ CCS-j
for each $c_l \in$ CCS-1
candidate = $X_k \cup \{c_l\}$
if |*candidate*| = $j+1$
Apply $(j+1)$-Column Counting Check on candidate
If the check is satisfied, CCS-$(j+1)$ = CCS-$(j + 1) \cup$ *candidate*
$j = j+1$
7. Suppose that the loop of Step 6 terminated at $j = k$.
Choose $A \in$ CCS-k
For each row l of RCOT
Let B is the entry in the S-column and row l
First k bits in encoding of B = entries in row l and columns$\in A$
8. Find set of symbols with the same encoding of the first k bits.
Assign a distinct combination of $(m - k)$ bits to members of the same set to complete the encoding.
9. end

Now, we illustrate Algorithm 1 with an example. We first apply Theorem 1 (Consistency Check) to the specification in Table 8. The symbolic output column (S-column) has the same value (X_2) in rows 2 and 4. But output column c_1 has a 1 in row 2 and a 0 in row 4. Thus, c_1 is inconsistent and need not be considered when determining the encoding of the symbols. Similarly, the symbolic output column has the same value (X_3) in rows 3 and 7 but output column c_3 has a 0 in row 3 and 1 in row 7. Hence, c_3 is also inconsistent. Thus we obtain the Consistent Output Table (COT) shown in Table 9(a) to which we apply Theorem 2 to obtain Reduced Consistent Output Table (RCOT) (Table 9(b)) and the Reduced Consistent Bound Output Table (RCBOT) (Table 9(c)).

Table 8. The Output Portion of a Specification Table

c_1	c_2	c_3	c_4	c_5	c_6	Symbolic
0	0	1	1	1	0	X_1
1	0	0	1	0	1	X_2
1	0	0	1	1	1	X_3
0	0	0	1	0	1	X_2
0	1	1	1	0	0	X_4
0	1	0	1	1	0	X_5
1	0	1	1	1	1	X_3
1	1	0	0	0	0	X_6
1	0	1	1	1	1	X_3
1	0	0	1	0	1	X_2

Table 9(a). The Consistent Output Table

c_2	c_4	c_5	c_6	Symbolic
0	1	1	0	X_1
0	1	0	1	X_2
0	1	1	1	X_3
0	1	0	1	X_2
1	1	0	0	X_4
1	1	1	0	X_5
0	1	1	1	X_3
1	0	0	0	X_6
0	1	1	1	X_3
0	1	0	1	X_2

Table 9(b). The RCOT

c_2	c_4	c_5	c_6	Symbolic
0	1	1	0	X_1
0	1	0	1	X_2
0	1	1	1	X_3
1	1	0	0	X_4
1	1	1	0	X_5
1	0	0	0	X_6

Table 9(c). The RCBOT

c_2	c_4	c_5	c_6
0	1	1	0
0	1	0	1
0	1	1	1
1	1	0	0
1	1	1	0
1	0	0	0

Now we apply Theorem 3 (Single Column Counting Check) to the columns of Table 10(b). Since there are six distinct symbols, $n = 6$ and $m = 3$ (because we are encoding using the minimum number of bits). Thus, the count of 1s (and 0s) in each column of Table 10(b) that pass the Single Column Counting check must not be more than 4. We find that columns c_2, c_5 and c_6 pass the Single Column Counting Check and hence the Column-Count-Set-1 (CCS-1) is $\{\{c_2\}, \{c_5\}, \{c_6\}\}$. Now we apply the 2-Column-Counting-Check (Theorem 4) to form CCS-2, the candidates being $\{c_2, c_5\}$, $\{c_2, c_6\}$ and $\{c_5, c_6\}$. Two of them satisfy the 2-Column Counting Check and hence the set CCS-2 is $\{\{c_2, c_5\}, \{c_5, c_6\}\}$. Next, we determine the candidates for CCS-3 by combining the members of CCS-2 and CCS-1. The only candidate for CCS-3 is $\{c_2, c_5, c_6\}$. But $\{c_2, c_5, c_6\}$ does not satisfy the 3-Column Counting Check (Theorem 4, $i = 3$) because 100 appears twice. Hence, CCS-3 is a null set, and for the given specification we can encode the symbols using three bits in such a way that two newly generated columns can be merged with the existing output columns when the output column cover is calculated. To determine the actual encoding of the symbols, we choose any member of CCS-2, say, $\{c_2, c_5\}$. The first two bits in the encoding of a particular symbol will have the same pattern as the one present under the columns c_2 and c_5 in the row corresponding to that symbol in the RCOT. The first two bits of all the symbols will not be distinct. Hence, we determine the third bit in such a way that the codes assigned to the symbols are distinct. Thus, referring to Table 9(a), X_1 will be encoded as 010, X_2 as 000, X_3 as 011, X_4 as 100, X_5 as 110 and X_6 as 101. Tables 10(a) and 10(b) show the final table (after encoding) before and after computing the output column cover.

Table 10(a). Encoded Symbols of Table 8

c_1	c_2	c_3	c_4	c_5	c_6	Encoding
0	0	1	1	1	0	010
1	0	0	1	0	1	000
1	0	0	1	1	1	011
0	0	0	1	0	1	000
0	1	1	1	0	0	100
0	1	0	1	1	0	110
1	0	1	1	1	1	011
1	1	0	0	0	0	101
1	0	1	1	1	1	011
1	0	0	1	0	1	000

Table 10(b). Column compaction on Table 10 (a)

c_1	c_2	c_3	c_4	c_5	c_6	c_7
0	0	1	1	1	0	0
1	0	0	1	0	1	0
1	0	0	1	1	1	1
0	0	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	1	0	0
1	0	1	1	1	1	1
1	1	0	0	0	0	1
1	0	1	1	1	1	1
1	0	0	1	0	1	0

The optimality of our algorithm can be proved very easily. As a contradiction, let us suppose that our algorithm does not produce the optimal encoding in terms of the cardinality of the cover of the output columns computed after encoding the symbols of the S-column. Specifically, let us suppose that our algorithm tells us that a maximum of *m* bits of the encoding can be merged with the elements of the B-set. But let us suppose that there exists an encoding E_2 such that $(m+1)$ encoding bits can be merged with the elements of the B-set. Let us call this subset of the B-set *S*. The size of *S* is $(m+1)$. Then, *S* must satisfy CCS-$(m+1)$. But our algorithm reported CCS-$(m+1)$ to be null. This means that either at least one element of *S* does not satisfy Theorem 3 or there exists a subset of size *m* of *S* which is not a member of CCS-m. In either case, these columns cannot be members of CCS-(m+1) because any subset of size *i* of an element of CCS-$(i+1)$ should be a member of CCS-i ($i \geq 1$).

A rough estimate of the time complexity of Algorithm 1 is described next. The complexity of step 1 is $r\log_2 r + rc$, where *r* is the number of rows in the given specification table and *c* is the cardinality of the B-set. The complexity of Step 3 is *r*. The complexity of step 5 is *kn* where *k* is the cardinality of the reduced B-set and *n* is the number of symbolic outputs. Computation of CCS-i has a worst case complexity of $\binom{k}{i}.2^i.n.i$. Thus, the net complexity of steps 5 and 6 is : $\sum_{i=1}^{m}\binom{k}{i}.2^i.n.i$, where *m* is the number of bits used to encode the symbolic outputs. Step 7 has a worst case complexity of $n\log_2 n + n$.

For the example of Table 10(a), to determine the third bit in the encoding of the symbolic outputs, a conventional output encoding algorithm (described in [Devadas 91]) could be used. For the above example, the first two bits of X_1 and X_3 are equal. Hence, we specify an output encoding problem using two symbolic outputs (and hence one-bit encoding) so that these two outputs are distinguished. Similar case holds for X_4 and X_6 as follows :

Table 10(c). Symbolic outputs for further minimization

c_1	c_2	c_3	c_4	c_5	c_6	Symbol
0	0	1	1	1	0	s_1
1	0	0	1	0	1	s_1
1	0	0	1	1	1	s_2
0	0	0	1	0	1	s_1
0	1	1	1	0	0	s_1
0	1	0	1	1	0	—
1	0	1	1	1	1	s_2
1	1	0	0	0	0	s_2
1	0	1	1	1	1	s_2
1	0	0	1	0	1	s_1

In Table 10(c), we distinguish between X_1 and X_3 by assigning s_1 to the rows corresponding to X_1 and s_2 to the rows corresponding to X_3. Similarly, s_1 has been assigned to the rows corresponding to X_4 and s_2 to the rows corresponding to X_6. In order to ensure that X_2 has a single code assigned to it, we have assigned s_1 to each row corresponding to X_2. Since, there is a single entry in the table corresponding to X_5, we have left in the entry corresponding to X_5 as don't care in the above table. However, further improvements can be done on this basic scheme as follows:

(a) If we assign 00- to X_2, then we could can allow don't care entries in the rows of Table 10(c), corresponding to X_2. This allows for additional optimization.

(b) Note that in Table 10(c), we could also serve our purpose by assigning s_2 to the rows corresponding to X_4 and s_1 to the rows corresponding to X_6. The complexity of the resulting logic will vary according to the order of this assignment. An interesting problem is to determine the order of this assignment so that the resulting logic is minimal. However, we do not address this problem in this report.

For Table 8, if we relaxed the constraint of encoding the symbolic outputs using the minimum number of bits (3 bits) and use 4 encoding bits, all the outputs corresponding to the encoding bits can be merged with the elements of the reduced B-set. In that case, X_1 will be encoded as 0110, X_2 as 0101, X_3 as 0111, X_4 as 1100, X_5 as 1110 and X_6 as 1000. Algorithm 1 can be extended to take care of this case, and is shown as Algorithm 1.A.

```
ALGORITHM 1.A

Steps :

1. Apply Steps 1 through 4 of Algorithm 1.
for (i = ⌈log2(symbolic output count)⌉; i ≤ min(symbolic output count, |reduced B-set|); i = i + 1) {
   Apply steps 5 and 6 of algorithm 1 using m = i.
   Find the largest j, CCS-j ≠ NULL.
   For each i, extra_i = i - j.
   If (j equals i)
       break;  /*** We can encode using i bits so that all the outputs corresponding to the encoding bits can
                be merged with the existing outputs ***/
  }
Choose the number of bits (p) for encoding the symbolic outputs such that extra_p is minimum.
```

In the next section, we extend our encoding algorithm to consider the case where the given specification may have *don't cares* in the columns of the B-set.

4. THE ENCODING ALGORITHM FOR OUTPUTS WITH DON'T-CARES

In this section, we extend the encoding algorithm described in Sec. 3 to support *don't cares* in the output portion of the specification table. The basic steps of the algorithm are the same as is described in Sec. 3. However, we make some extensions to Theorems 1, 2, and 4 (calling them Theorems 1-DC, 2-DC, and 4-DC, respectively) in order to consider *don't cares*.

Theorem 1-DC : Check Consistency with Don't-Cares

If there exist two rows *p* and *q* in the truth table (or the state table) for which the S-column is specified and has the same value and $c_i \in$ B-set has specified but different values (i.e., c_i does not have a *don't care* in row *p* or *q*), then c_i cannot contribute to reducing the cardinality of the output column cover after encoding the S-column.

Proof : The proof of Theorem-1-DC is exactly the same as that of Theorem 1.

Applying Theorem 1-DC, we can obtain the reduced B-set and the consistent output table (COT). Next, we apply Theorem 2-DC to obtain the reduced consistent output table (RCOT).

Theorem 2-DC : Compatible Row Merging

In the consistent output table, any two rows having the same values for the S-column are compatible and can be merged. While merging compatible rows, *don't cares* should be fixed to 0s or 1s whenever necessary.

Proof : The proof is straightforward and follows the proof of Theorem 2. However, special care must be taken while merging compatible rows. For example, if row *p* is merged with row q and column x in the COT has a 1(0) in row p and a - (*don't care*) in row q, the merged row must have a 1(0) in column x. Since the columns of the COT satisfy the Consistency Check (Theorem 1-DC), no case can arise for which it is required to change the same *don't care* to 0 for one particular compatible row and to a 1 for another compatible row. Q.E.D.

By repeated application of Theorem 2-DC for each symbol of the symbolic output column, we form the RCOT and the RCBOT as described in the Sec. 3. However, the RCOT and the RCBOT may contain *don't cares* in this case.

The Single Column Counting Check (Theorem 3) holds for the RCBOT with *don't cares* ; this is because, once the number of 1s and 0s in a particular column is within the specified upper limit, there always exists a satisfying assignment of 1s and 0s to the *don't cares* in that column so that the upper limit on the count of 0s and 1s is maintained. However, there is a minor modification in the i-Column Counting Check (Theorem 4) which we present below as Theorem 4-DC.

Theorem 4-DC : *i* - Column Counting check with Don't Cares

Any set of i columns of RCBOT having any of the 2^i possible binary combinations formed by the entries under those columns in each row of RCBOT which are fully specified (i.e., there is no *don't care* in that row under those i columns), appearing more than 2^{m-i} times cannot help to reduce the size of the output column cover by i after encoding the symbols. Here m is the number of bits used to encode the symbols and to encode using the minimum number of bits, $m = \lceil \log_2 n \rceil$, where n is the number of symbols to be encoded (number of rows of RCBOT).

Proof: The proof of Theorem 4-DC is the same as the proof of Theorem 4.

As mentioned previously, we must consider the count of those strings under the i columns which are fully specified. This is because if the fully specified strings satisfy the upper bound, there always exists a satisfying assignment of 1s and 0s to the *don't cares* so that the upper bound is not violated. As described in the Sec. 3, we construct a family of sets CCS-i starting from i equal to 1 and stopping when either CCS-i is empty or i is greater than the number of bits required to encode the symbols in the S-column.

Let us suppose that the computation of the CCSs terminates with CCS-j. For the incompletely specified case, we select any member of CCS-j, and for each row of the RCBOT, we determine a satisfying assignment of 0s and 1s to the *don't care* entries in the selected j columns so that after the assignment, the condition in Theorem 4 (j-Column

Counting Check) is not violated. We solve this problem by modeling it as a maximum network flow problem and solving it by using the Ford-Fulkerson method [Cormen 89].

A *bipartite graph* is a graph $G = (V, E)$ in which the vertex set V can be partitioned into two non-trivial disjoint subsets V_1 and V_2 such that $(u, v) \in E$ implies that $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$. [Cormen 89]. For our purpose, each member of V_1 represents a symbol and has a label which the string present in the selected j columns in the row corresponding to that symbol in the S-column of RCOT. The set V_2 contains 2^j elements, each element representing a distinct binary pattern of length j. For each $u \in V_1$ and $v \in V_2$ we have an edge (u, v) if and only if u covers v. u is said to *cover* v if:

(i) u and v are bitwise equal or

(ii) v has 1 (0) in all bit positions in which u has 1 (0). (e.g., 0-11-10 covers 011110, 0011010, 0011110 and 0111010).

For each edge, we add another attribute called the *weight* of the edge. All these edges have a weight of 1. We introduce two more vertices, *source* and *sink*, to the above graph and add edges from the *source* to all members of V_1, each with weight 1, and to the *sink* from all members of V_2, each with weight 2^{m-j}. Note that, here m is the number of bits used to encode the symbolic outputs and j corresponds to the CCS-j under consideration. We solve the maximum network flow problem for the resulting network (the edge weights indicate the corresponding capacities) using Ford-Fulkerson's method [Ford 62][Cormen 89]. Since the weight (capacity) of each edge from *source* to an element of V_1 is 1 and the weight (capacity) of each edge between V_1 and V_2 is 1, we ensure that each node of V_1 gets mapped to one and only one node of V_2. Moreover, since the weight of an edge between an element of V_2 and *sink* is 2^{m-j}, a maximum of 2^{m-j} elements of V_1 can be mapped to a particular node in V_2 thereby satisfying the constraints imposed by Theorem 4. In the maximum network flow problem, given a graph (network) with the edge weights representing the capacities of the links between two nodes, we determine the maximum flow that can be achieved between the *source* and the sink node. It can be easily proved that the problem of mapping an element of V_1 to one and only one element of V_2 can be modeled as a network flow problem. The proof closely follows the proof of computing *Maximum Bipartite Matching* given in [Cormen 89]. After solving the maximum flow problem, let us consider the set S of edges between V_1 and V_2 in the graph with non-zero flow. For any edge $(u, v) \in S$, $u \in V_1$ and $v \in V_2$, we can map u to v without violating the condition given by j-Column Counting check (Theorem 4). The subsequent encoding of the remaining $(m-j)$ bits is the same as in Algorithm 1.

Now, we report our algorithm and illustrate the entire procedure with the help of an example.

ALGORITHM 2: Encode Symbols for an Output Table With Don't Cares

Input : The output portion of the given specification table; one column is for symbolic output and the other columns are contain 1s, 0s and *don't cares*.

Output : An encoding of the symbols so that the cardinality of the output column cover computed after encoding the symbols is minimum.

Steps :

1. **Apply Theorem 1-DC** similar to **Algorithm 1**
2. Form the **Consistent Output Table** (COT) consisting of the *consistent* columns and the **S-Column**.
3. **Apply Theorem 2-DC** similar to Algorithm 1
4. Calculate the number of rows of RCOT. Let it be n. Let $m = \lceil \log 2n \rceil$.
5. **Apply Theorem 3** similar to **Algorithm 1**
6. **Apply Theorem 4-DC** similar to Step 6 of **Algorithm 1**
7. Suppose that the loop of Step 6 terminated at $j = k$.
Choose a member of CCS-k, say, A
Create a bipartite graph G as follows :
(a) Set V_1 consists of vertices corresponding to each row of the RCOT.
The label of each such vertex is the string formed by the entries in that row corresponding to the columns which are elements of A
(b) Set V_2 consists of 2^k vertices each representing a binary combination of k bits.
Add edge (u, v) of weight 1, $u \in V_1$ and $v \in V_2$, if and only if the label of u covers the binary combination that v represents.
Create a node *SOURCE* and add edges of weight 1 from *SOURCE* to each element of V_1.
Create a node *SINK* and add edges of weight $2^{(m-k)}$ from each element of V_2 to *SINK*.
8. Solve the maximum flow problem for G using Ford-Fulkerson's method.
9. Find out edges (u, v) in G, $u \in V_1$ and $v \in V_2$, with flow$(u, v) \neq 0$
Encode first k bits of symbol u same as the vector represented by v.
10. Find out sets of symbols with the same encoding of the first k bits.
Assign a distinct combination of $(m - k)$ bits for members of the same set.
11. end

Table 11. The Output Portion of a Specification Table

c_1	c_2	c_3	c_4	c_5	c_6	Symbolic
0	—	1	0	—	0	X_1
1	0	—	—	—	—	X_2
1	—	—	—	1	—	X_3
0	—	—	—	—	1	X_2
0	1	0	0	0	0	X_4
0	1	1	0	—	—	X_5
—	—	1	1	—	—	X_3
1	1	1	0	0	0	X_6
—	—	—	1	1	—	X_3
—	—	1	0	—	—	X_2

Table 12. The Consistent Output Table

c_2	c_3	c_4	c_5	c_6	Symbolic
—	1	0	—	0	X_1
0	—	—	—	—	X_2
—	—	—	1	—	X_3
—	—	—	—	1	X_2
1	0	0	0	0	X_4
1	1	0	—	—	X_5
—	1	1	—	—	X_3
1	1	0	0	0	X_6
—	—	1	1	—	X_3
—	1	0	—	—	X_2

To illustrate our algorithm, let us consider the output portion of an example specification, shown in Table 11. Applying Theorem 1-DC to Table 11, we find that the output column c_1 does not satisfy the consistency check with *don't cares* because the column has a 1 in row 2 and a 0 in row 4, whereas the symbol corresponding to row 2 and row 4 is X_2. However, the other output columns satisfy the consistency constraint mentioned in Theorem 1-DC. Table 12 is the Consistent Output Table (COT) obtained after applying Theorem 1-DC to Table 11. We apply the reduction procedure in Theorem 2-DC to obtain the RCOT and the RCBOT shown in Tables 13(a) and 13(b) respectively.

Table 13(a). The Reduced Consistent Output Table

c_2	c_3	c_4	c_5	c_6	Symbolic
—	1	0	—	0	X_1
0	1	0	—	1	X_2
—	1	1	1	—	X_3
1	0	0	0	0	X_4
1	1	0	—	—	X_5
1	1	0	0	0	X_6

Table 13(b). The RCBOT of Table 12

c_2	c_3	c_4	c_5	c_6
—	1	0	—	0
0	1	0	—	1
—	1	1	1	—
1	0	0	0	0
1	1	0	—	—
1	1	0	0	0

Now we apply the Single Column Counting Check, as per Theorem 3, to the columns of the RCBOT of Table 13(b). Since the number of distinct symbols (number of rows of RCBOT) is six, m, the number of bits required to encode the symbols, is 3. We find that column c_3 has more than four 1s while column c_4 has more than four 0s. Hence, the CCS-1 set is $\{\{c_2\}, \{c_5\}, \{c_6\}\}$. Now we apply 2-Column Counting Check according to Theorem 4-DC to the possible candidates of CCS-2 viz. $\{c_2, c_5\}$, $\{c_2, c_6\}$ and $\{c_5, c_6\}$. We find that all of them satisfy the counting check. Finally, we try the 3-Column Counting Check (Theorem 4-DC, i = 3) on $\{c_2, c_5, c_6\}$. We find that *100* appears twice in row 4 and row 6 of Table 13(b) and hence violates Theorem 4-DC for i = 3. Hence, CCS-3 is a null set and we choose $\{c_2, c_5\}$, a member of CCS-2, for providing the first two bits in the encoding of the symbols using 3 bits.

Next, we form the graph described previously to model the problem of finding a satisfying assignment of 0s and 1s to the *don't cares* of the columns c_2 and c_5 so that the constraint imposed by Theorem 4 for i = 2 is satisfied. The set V_1 contains six nodes representing the six symbols X_1, X_2, X_3, X_4, X_5 and X_6 with labels --, 0-, -1, 10, 1- and 10, respectively. The set V_2 contains four elements 00, 01, 10 and 11. The graph along with the source and the sink nodes and the edge weights is shown in Fig. 4. We solve the maximum network flow problem on the graph of Fig. 4, and the solid edges show the final mapping (edges corresponding to the non-zero flow). Thus, the first two bits in the encoding of X_1, X_2, X_3, X_4, X_5 and X_6 are 00, 01, 01, 10, 11 and 10, respectively. Hence, the encodings of X_1, X_2, X_3, X_4, X_5 and X_6 are 001, 010, 011, 100, 111 and 101, respectively. As shown in Fig. 4, there is more than one possible mapping — hence, there can be multiple encodings of the symbols. Our algorithm can be further refined by selecting a mapping based on some heuristic developed on the basis of observations in [Devadas 91] and [Saldanha 88]. Table 14(a) shows the table obtained after encoding and Table 14(b) shows the table obtained after applying output column compaction to Table 14(a).

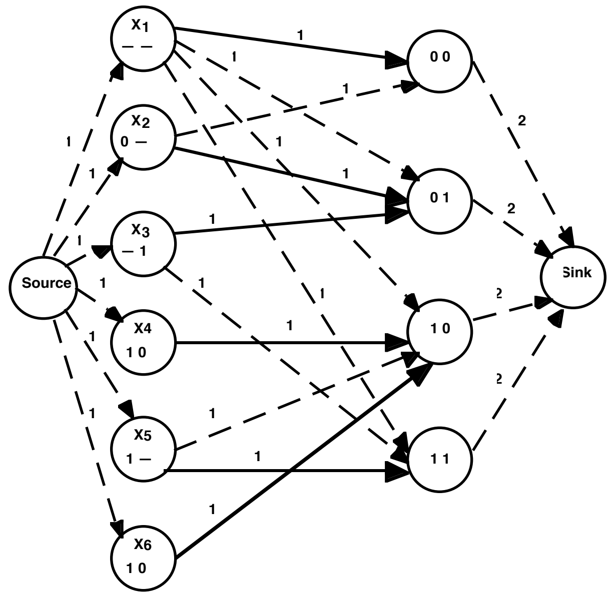

Figure 4. The Graph for the Maximum Flow Problem

Table 14(a). Encoding of the symbols

c_1	c_2	c_3	c_4	c_5	c_6	Encoding
0	0	1	0	0	0	001
1	0	—	—	1	—	010
1	0	—	—	1	—	011
0	0	—	—	1	1	010
0	1	0	0	0	0	100
0	1	1	0	1	—	111
—	0	1	1	1	—	011
1	1	1	0	1	0	111
—	0	—	1	1	—	011
—	0	1	0	1	—	010

Table 14(b). Column compaction on Table 14(a)

c_1	c_2	c_3	c_4	c_5	c_6	c_7
0	0	1	0	0	0	1
1	0	—	—	1	—	0
1	0	—	—	1	—	1
0	0	—	—	1	1	0
0	1	0	0	0	0	0
0	1	1	0	1	—	1
—	0	1	1	1	—	1
1	1	1	0	1	0	1
—	0	—	1	1	—	1
—	0	1	0	1	—	0

5. EXPERIMENTAL RESULTS

In this section, we present some experimental results. We added a symbolic output column to the MCNC FSM benchmarks and varied the number of symbols in the symbolic output column; depending on the symbol count, the number of encoding bits required were 3 and 4. We entered symbolic values in the S-column to ensure that there exists an output encoding for which all the columns generated due to the encoding of the symbols could be merged with the members of the B-set, assuming that the number of encoding bits did not exceed the number of existing non-symbolic (binary) output columns. We applied our output encoding algorithm to encode the symbolic output columns, then compacted the outputs using column compaction. We then optimized these specifications using *sis* [Sentovich 92]. For comparison purposes, we then encoded the symbolic outputs such that none of the outputs corresponding to the encoding bits could be merged by column compaction with the pre-existent outputs (members of the B-set). This is the worst-case encoding. The results are shown in Table 15. Although our scheme works for both combinational and sequential logic specifications, we used FSM benchmarks because we are investigating FSM synthesis techniques for one-hot signals, as mentioned in Sec. 2 [Mitra 97a]. For both cases, we used NOVA [Villa 90] to encode the FSM states and used the recommended *rugged* script to perform multi-level logic optimization. Finally, we used the LSI Logic g10p library [LSI 96] for technology mapping. Table 15 shows the area values (in terms of LSI Logic g10p cell units) obtained using our output encoding algorithm and the worst-case output encoding where none of the encoding bits could be merged with the output columns of the B-set by compaction.

Table 15. Experimental results : Area comparisons using our output encoding algorithm (best case) with the output encoding algorithm which never merges any output column (Worst Case)

FSM	3 bits for encoding		4 bits for encoding	
Name	Our algo.	Worst Case	Our algo.	Worst Case
bbara	*205	225	*285	346
bbsse	316	364	316	378
bbtas	*98	99	*103	147
dk14	223	266	223	294
dk15	206	208	206	257
dk17	*182	191	*231	304
ex1	721	750	721	781
ex6	254	312	254	405
mc	93	96	93	96
opus	235	260	235	356
tav	73	143	73	188
tma	444	543	444	548

* : These are the cases where all the output columns could not be merged in the best case because the cardinality of the B-set is less than the number of bits needed to encode the symbols.

6. EXTENSION TO ELEMENTARY GATES

The scheme for performing output encoding, described in sections 3 and 4 can be extended in the following way. Let us define S as a set of *elementary* gates. S may consist of two or three input AND and OR gates. Note that the inverter need not be considered in S because it has already been considered implicitly in Sec. 3 and 4. For the purpose of a simple example, let us consider that S consists of a two-input AND gate. If the number of elements of the B-set is n, we generate $n(n-1)/2$ extra output columns each of which is the result of AND-ing a pair of elements of the *B-set*. For two output columns i and j belonging to the *B-set*, we generate the output column $AND_{i,\ j}$ in the following way:

For each row of the given table, if the entry in column i or j is 0, then the corresponding entry in column $AND_{i,\ j}$ is 0; if both the entries in columns i and j are 1, then the entry in column $AND_{i,\ j}$ is 1; otherwise, the entry in column $ANDi,\ j$ is a *don't care*. In general, if one of the entries in column i or j determines the output of the gate under consideration (*controlling value*), then the entry in the new column is the resulting *controlled value* (output of the gate); if the entries in columns i and j are fully specified then the entry in the new column is the result of applying the values to the gate under consideration; otherwise, the entry in the newly generated column is a don't care. We call these newly generated output columns *derived output columns*. All derived output columns together with the original set of output columns, form the *Extended-Bound Set* (EB-Set). The necessity of the EB-Set is explained with the help of the following example.

Table 16. Use of Elementary gates to generate extra output columns

c_1	c_2	Symbol	$AND_{1,\ 2}$
0	0	s_1	0
1	0	s_1	0
1	1	s_2	1
0	0	s_1	0
0	1	s_1	0
1	1	s_2	1
1	1	s_2	1
1	1	s_2	1
1	0	s_1	0

Let us consider the output part of a very simple specification, shown in Table 16, that has a single symbolic output column and two elements in the B-Set, c_1 and c_2. The symbolic output has two values, s_1 and s_2, and hence, a single bit is sufficient to encode the symbolic output. Neither c_1 nor c_2 are consistent because c_1 has a 0 in row 1 and a 1 in row 2 while the symbolic column has s_1 in both of these two rows; column c_2 has a 0 in row 1 and a 1 in row 5 while both of these two rows have s_1 under the symbolic output column. However, if we considered the output column $AND_{1,\ 2}$ obtained by AND-ing the output columns c_1 and c_2, then we find that $AND_{1,\ 2}$ is a consistent column and can be used to encode s_1 and s_2. In this case, we encode s_1 as 0 and s_2 as 1. The general framework that we developed in Sections 3 and 4 can now be used to solve the problem. However, we have an additional optimization step. In the final set of columns (which are elements of the EB-Set), we want to maximize the number of the originally present output columns (the members of the B-Set) and minimize the number of derived output columns. This can be done in the following way: when we calculate the CCS-i sets and reach a point when *CCS-j* is NULL, our algorithm in Sec. 3 and 4 picked up any element of *CCS-(j-1)* for performing the actual encoding; instead of that, now we choose the element of the *CCS-(j-1)* set which has the minimum number of derived columns.

7. CONCLUSION

In this report, we have presented a new output encoding algorithm whose objective is to encode the symbols in the symbolic output column of the specification table in such a way that the number of output functions, after performing the encoding and subsequent output column compaction, is minimum. An application of this algorithm is to generate a low-area circuit that always maintains a one-hot encoding on certain signals, even during scan or BIST operations, as discussed in [Mitra 97a]. This algorithm can also be used as a pre-processing step for FSM state encoding. We have proved the correctness of the algorithm and analyzed the worst case time complexity of the algorithm. Although the algorithm produces a minimum solution and the worst case running time is exponential in the number of bits used to encode the symbolic outputs, our algorithm achieves significant speedup through iterative refinement by removing from consideration inconsistent output columns or sets of output columns that do not satisfy the conditions in Theorems 1 (DC), 3 and 4 (DC). Experimental results show that the circuits generated using our output encoding algorithm have significantly less area (0.9 % to 49 % for 3 bits, 8 % to 61 % for 4 bits for the example circuits we considered) than worst-case circuits generated without considering output column compaction during the encoding of the symbols. In this report, we have considered specifications with a single symbolic output column. We also considered the extension of our algorithm for functions realizable using elementary gates. For multiple symbolic output columns, we can integrate our technique with the output encoding algorithm reported in [Buijs 91] to achieve a minimum number of output columns in the final table.

8. ACKNOWLEDGEMENTS

The authors wish to thank Nirmal Saxena, Robert Norwood and Jonathan Chang for their comments and suggestions. This work was supported by the Advanced Research Projects Agency under prime contract No. DABT63-94-C-0045.

9. REFERENCES

[Ashar 91] Ashar, P., S. Devadas and A. R. Newton, *Sequential Logic Synthesis*, Kluwer Academic Publishers, USA, 1991.

[Binger 91] Binger, D. and D. W. Knapp, “Encoding Multiple Outputs for Improved Column Compaction,” *Proc. ICCAD-91*, pp. 230-233, 1991.

[Brayton 84] Brayton, R. K., G. D. Hachtel, C. T. McMullen and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[Buijs 91] Buijs, F. and T. Lengauer, "Synthesis of Multi-Level Logic with one symbolic input," *EDAC-91*, pp. 60-64, 1991.

[Cormen 89] Cormen, T. H., C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, The MIT Press and McGraw-Hill Book Company, 1989.

[De Micheli 84] De Micheli, G., R. Brayton and A. Sangiovanni-Vincentelli, "KISS: A Program for Optimal State Assignment of Finite State Machines," *Proc. ICCAD*, pp. 209-211, 1984.

[De Micheli 85] De Micheli, G., R. Brayton and A. Sangiovanni-Vincentelli, "Optimal State Assignment for Finite State Machines," *IEEE Trans. on CAD.*, CAD-4(3), 269-285, July 1985.

[Devadas 91] Devadas, S. and A. R. Newton, "Exact Algorithms for Output Encoding, State Assignment and Four Level boolean Minimization," *IEEE Trans. on CAD*, 10(1), pp. 13-27, Jan. 1991.

[Dolotta 64] Dolotta, T. A., and E. J. McCluskey, "The Coding of Internal States of Sequential Circuits," *IEEE Trans. Comput.*, EC-13, pp. 549-562, Oct. 1964.

[Du 91] Du, X., G. Hachtel, B. Lin and A. R. Newton, "MUSE: A MUltilevel Symbolic Encoding Algorithm for State Assignment," *IEEE Trans. on CAD*, 10(1), pp. 28-38, January 1991.

[Ford 62] Ford, L. R. and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.

[Lin 90] Lin, B. and A. R. Newton, "Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages," *VLSI 89*, pp. 187-196, Elsevier Science Publishers, 1990.

[LSI 96] *G10-p Cell-Based ASIC Products Databook*, LSI Logic, May 1996.

[McCluskey 59] McCluskey, E. J. and S. H. Unger, "A Note on the Number of Internal Variable Assignments for Sequential Switching Circuits," *IRE Trans. Electron. Comput.* , EC-8, pp. 439-440, Dec. 1959.

[McCluskey 86] McCluskey, E. J., *Logic Design Principles with Emphasis on Testable Semicustom circuits*, Prentice-Hall, Eaglewood Cliffs, NJ, USA, 1986.

[Mitra 97a] Mitra, S., L. J. Avra and E. J. McCluskey, "Scan Synthesis for One-hot Signals", *Proc. International Test Conference*, pp. 714-722, 1997.

[Mitra 97b] Mitra, S., L. J. Avra and E. J. McCluskey, "An Output Encoding Problem and a Solution Technique", *Proc. ICCAD-97,* pp. 304-307, 1997.

[Saldanha 88] Saldanha, A. and R. H. Katz, "PLA Optimization Using Output Encoding," *Proc. ICCAD-88,* pp. 478-481, 1988.

[Sentovich 92] Sentovich, E., *et. al.*, "SIS: A System for Sequential Circuit Synthesis", *Electronics Research Laboratory Memo. No. UCB/ERL M92/41*, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

[Tumbush 74] Tumbush, G. L. and J. E. Brandeberry, "A State Assignment Technique for Sequential Machines using J-K Flip-Flops," *IEEE Trans. Comput.*, pp. 85-86, Jan. 1974.

[Villa 90] Villa, T. and A. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementation", *IEEE Trans. on CAD*, 9(9), pp. 905-924, Sept. 1990.

[Wei 87] Wei, R. and C Tseng, "Column Compaction and Its Application to The Control Path Synthesis," *Proc. ICCAD-87*, pp. 320-323, 1987.

[Yang 91] Yang, S. and M. J. Ciesielski, "Optimum and Suboptimum Algorithms for Input Encoding and Its Relation to Logic Minimization," *IEEE Trans. on CAD*, 10(1), pp. 4-12, Jan. 1991.