PRELIMINARY VERSION

Center for Reliable Computing TECHNICAL REPORT

An Output Encoding Problem And A Solution Technique

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

99-1

(CSL TR # 00-794)

May, 1999

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. We are 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. The number of bits for encoding the output symbols may also be specified. We have to 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 technique, 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 simple logical combinations of the already specified output columns to facilitate further reduction in the number of output functions.

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. 99-1
(CSL TR No. 00-794)
May 1999

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. We are 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. The number of bits for encoding the output symbols may also be specified. We have to 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 technique, 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 simple logical combinations of the already specified output columns to facilitate further reduction in the number of output functions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Column compaction plays a major role in the synthesis of digital systems [22]. 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 each input combination, 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 the 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 *Graph Coloring Problem*, which is an NP-Complete problem [22]. 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 [3] and for multi-level implementation of logic circuits.

Figure 1. Examples of column compaction.

Encoding problems include the *finite state machine (FSM) state encoding* problem and the *input* and the *output* encoding problems. 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 [14]. Techniques for FSM state encoding have been discussed in [1] [6] [7] [9] [10] [12] [20] [21]. The input encoding problem is discussed in [4] and [23]. One version of the output encoding problem to reduce the number of product terms of a logic function is discussed in [8]. Another version of the output encoding problem is discussed in [18]. 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 [2]. 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 paper, we consider 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 with a specified number of encoding bits, so that after encoding and subsequent column compaction, the total number of output functions to be implemented is minimum. There are various applications of this output encoding problem. It can be used as a pre-processing step during conventional output and FSM state encoding. This technique also helps to reduce the area-overhead associated with a design for testability technique in the presence of one-hot signals. In this paper, we have presented experimental results showing the reduction in area obtained by using our algorithm during output encoding. In fact, our algorithm can be used for merging the output functions generated out of an encoding with the inputs of the given specification.

An outline of the algorithm was first presented in [17]. We present a more detailed description of the algorithm, including rigorous proofs, here. 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 outputs. In Sec. 4, we extend the algorithm given in Sec. 3 to handle *don't cares* in the outputs. Experimental results are reported in Sec. 5. In Sec. 6, we extend our algorithm, presented in Sec. 3 and 4, to handle basic gates. This leads to further reduction in the number of output functions by allowing merging of the output columns generated by the encoding process with the output functions generated by the basic gates. 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 b_1, b_2, b_3 and b_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 paper, 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 paper.

Table 1. Example Truth table with symbolic output.

Inputs	Outputs	
	Bound $b_1\ b_2\ b_3\ b_4$	Symbolic
10101	1 0 1 0	Z_1
01100	0 0 1 0	Z_2
10001	1 0 0 1	Z_3
01111	- 1 - 0	Z_2
11110	0 - 0 -	Z_4
01010	1 1 - 0	Z_5
11111	1 0 - 0	Z_1
1110-	0 - 1 1	Z_6
11011	1 0 1 1	Z_7
01001	0 1 0 1	Z_4

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 is $\{b_1, b_2, b_3, b_4\}$. 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 correspond to the encoding bits. 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 2. An encoding of the symbolic outputs of Table 1. (a) Encoding. (b) Truth table.

(a)

Signals	Encoding $c_5\ c_6\ c_7$
Z_1	100
Z_2	111
Z_3	101
Z_4	011
Z_5	010
Z_6	110
Z_7	001

(b)

Input	Outputs $b_1\ b_2\ b_3\ b_4\ \mathbf{c_5\ c_6\ c_7}$
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 encode 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), the encoding bit columns are merged with columns b_1, b_3 and b_4.

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. Encoding of symbolic outputs of Table 1. (a) The encoding. (b) The Truth table.

(a)

Signals	Encoding
Z_1	110
Z_2	010
Z_3	101
Z_4	001
Z_5	100
Z_6	011
Z_7	111

(b)

Input	Output **b_1** b_2 **b_3 b_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 [15], *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. Now, according to the flow shown in Fig. 2(b), it doesn't matter how we encode these one-hot signals using two bits, as long as we decode them correctly using the decoder. Hence, we can specify the four one-hot signals symbolically and find an efficient encoding so that the output logic of the FSM is minimized. The corresponding specification table with the one-hot signals replaced by symbolic outputs 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 detail in [16]. 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 micro-codes in order to reduce the width of micro-control memories. This algorithm can also be used as a pre-processing step during conventional output encoding [8] and FSM state encoding.

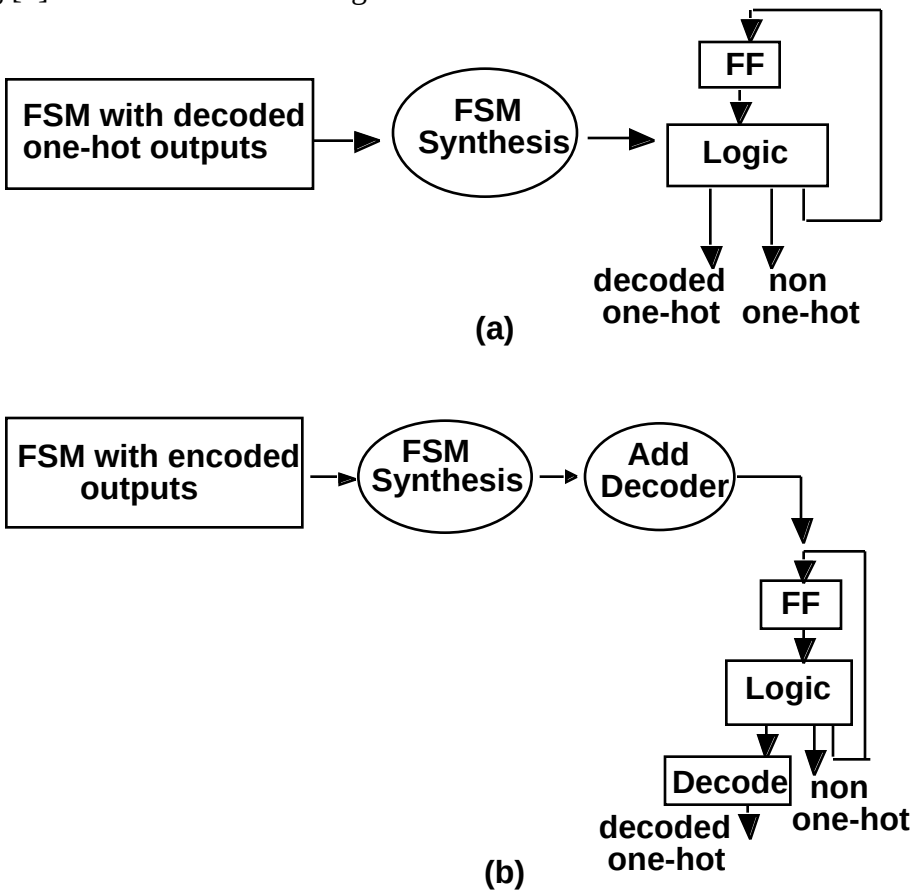

Figure 2. FSM synthesis techniques for one-hot signals. (a) Conventional synthesis scheme. (b) Scheme to ensure one-hot condition on control signals during scan or BIST.

Table 4. Illustration of the output encoding problem for FSMs producing one-hot signals. (a) Output part of FSM generating one-hot signals. (b) Symbolic outputs corresponding to one-hot signals of Table 4(a).

(a)

FSM Outputs	
$b_1\ b_2\ b_3\ b_4$	$c_5\ c_6\ c_7\ c_8$
- 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

(b)

Bound	Symbolic
$b_1\ b_2\ b_3\ b_4$	Output
- 0 1 -	z_1
- 0 1 0	z_2
0 1 - 1	z_4
1 - - 1	z_2
1 - - 1	z_3
0 0 1 0	z_1
- 1 0 -	z_3
1 1 - -	z_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 Z_j and b_i B-set has different values, then b_i cannot contribute to reducing the number of output columns after encoding the S-column. In this case, b_i is said to be *inconsistent* with respect to Z_j.

Proof : Without loss of generality, let us assume that b_i has a '1' in the row *p* and a '0' in the row *q* and the S-column contains Z_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 b_i can never match any column generated by any encoding of the output symbols. Hence, b_i cannot help in reducing the output column cover after encoding the S-column. Q.E.D.

We apply Theorem 1 to each b_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. For example, consider the output portion of a specification shown in Table 5. The B-set is $\{b_1, b_2, b_3, b_4, b_5, b_6\}$.

Table 5. The Output Portion of a Specification Table.

b_1	b_2	b_3	b_4	b_5	b_6	Symbolic
0	0	1	1	1	0	Z_1
1	0	0	1	0	1	Z_2
1	0	0	1	1	1	Z_3
0	0	0	1	0	1	Z_2
0	1	1	1	0	0	Z_4
0	1	0	1	1	0	Z_5
1	0	1	1	1	1	Z_3
1	1	0	0	0	0	Z_6
1	0	1	1	1	1	Z_3
1	0	0	1	0	1	Z_2

The symbolic output column (S-column) has the same value (Z_2) in rows 2 and 4. But output column b_1 has a 1 in row 2 and a 0 in row 4. Thus, b_1 is inconsistent and need not be considered when determining the encoding of the symbols. Similarly, the symbolic

output column has the same value (Z_3) in rows 3 and 7 but output column b_3 has a 0 in row 3 and 1 in row 7. Hence, b_3 is also inconsistent. The reduced B-set is {b_2, b_4, b_5, b_6}.

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). The COT for Table 5 is shown in Table 6. We can perform row merging on this consistent output table to obtain a *reduced consistent output table* (RCOT) using Theorem 2.

Table 6. The Consistent Output Table for Table 5.

b_2	b_4	b_5	b_6	Symbolic
0	1	1	0	Z_1
0	1	0	1	Z_2
0	1	1	1	Z_3
0	1	0	1	Z_2
1	1	0	0	Z_4
1	1	1	0	Z_5
0	1	1	1	Z_3
1	0	0	0	Z_6
0	1	1	1	Z_3
0	1	0	1	Z_2

Theorem 2 : Row Merging

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

Proof : The proof is straightforward. Consider any two rows *p* and *q* of the COT having the same value in the S-Column. Since any column (except the S-column) of the COT is a member of the reduced B-set, by Theorem 1, any other column of the 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.

Table 7. The RCOT and RCBOT for the COT of Table 6. (a) The RCOT. (b) The RCBOT.

(a)

b_2	b_4	b_5	b_6	Symbolic
0	1	1	0	Z_1
0	1	0	1	Z_2
0	1	1	1	Z_3
1	1	0	0	Z_4
1	1	1	0	Z_5
1	0	0	0	Z_6

(b)

b_2	b_4	b_5	b_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

By applying Theorem 2 to the COT for each value of the symbols in the S-column, we obtain the RCOT from the COT. Let the number of rows in the RCOT be *n*. This means that there are *n* distinct symbols in the S-Column which can be encoded using $m = \lceil \log_2(n) \rceil$ bits. Next, we remove the S-Column from the RCOT to obtain the

Reduced Consistent-Bound Output Table (RCBOT). The RCOT and the RCBOT for the COT of Table 6 are shown in Table 7(a) and Table 7(b), respectively. Once we obtain the RCBOT, we attempt to reduce the number of columns in the RCBOT using Theorem 3.

Theorem 3 : Single Column Counting Check

Any column of the RCBOT having more than 2^{m-1} 1s (or 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 ($1 \le i \le m$) 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. This is because, suppose that column e_1 contains ($2^{m-1} + 1$) 1s. Now, for the remaining $(m-1)$ bits ($e_2, ..., e_m$), there can be only 2^{m-1} distinct combinations. Hence, there will always exist two binary vectors v_1 and v_2 which will have identical entries for these $(m-1)$ bits. Now, these two vectors have 1 in bit e_1. Therefore, v_1 and v_2 are identical, i.e., two distinct symbolic outputs are assigned the same binary code. Hence, any column of the 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 ($1 \le i \le m$) and hence cannot help in reducing the size of the output column cover. Q.E.D.

Now we apply Theorem 3 to the columns of Table 7(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 7(b) that passes the Single Column Counting check must not be more than 4. We find that columns b_2, b_5 and b_6 pass the Single Column Counting Check. We call the set of columns satisfying Theorem 3, the *Column-Count-Set-1* (CCS-*1*). The CCS-*1* for the current example is $\{\{b_2\}, \{b_5\}, \{b_6\}\}$. 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).

Theorem 4 : *i* - Column Counting Check

Any set of i columns of the 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 the 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 sets of i columns that satisfy the i-Column Counting check form elements of CCS-i. 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 $(i + 1)$.

Now we apply the 2-Column-Counting-Check (Theorem 4) to the RCBOT in Table 7(b) and form CCS-*2*, the candidates being $\{b_2, b_5\}$, $\{b_2, b_6\}$ and $\{b_5, b_6\}$. Two of them satisfy the 2-Column Counting Check and hence the set CCS-*2* is $\{\{b_2, b_5\}, \{b_5, b_6\}\}$. 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 $\{b_2, b_5, b_6\}$. But $\{b_2, b_5, b_6\}$ does not satisfy the 3-Column Counting Check (Theorem 4, $i = 3$) because 100 appears twice in columns b_2, b_5 and b_6 in Table 7(b) (in rows 4 and 6). Hence, CCS-*3* is a null set.

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. For our current example, j is 2 and hence, for Table 5, we can encode the output symbols using three bits such that two encoding bit columns can be merged with the existing columns during compaction.

In general, we can encode the first j bits of the symbol in the S-column and row k of the RCOT with the binary values from row k of the RCOT in the columns that are members of CCS-j. For our current example, we arbitrarily choose a member of CCS-*2*, say, $\{b_2, b_5\}$. The first two bits in the encoding of a particular symbol will have the same pattern as the one present under the columns b_2 and b_5 in the row corresponding to that symbol in the RCOT (Table 7(a)). Thus, the first two bits in the encoding of Z_1 will be 01, that of Z_2 will be 00 and so on. 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.

For our example, the first two bits of all the symbols are not distinct. Hence, we determine the third bit in such a way that the codes assigned to the symbols are distinct. Thus, referring to Table 7(a), Z_1 will be encoded as 010, Z_2 as 000, Z_3 as 011, Z_4 as 100,

Z_5 as 110 and Z_6 as 101. Tables 8(a) and 8(b) show the final table (after encoding) before and after computing the output column cover.

Table 8. Output encoding. (a) Table 5 with encoded outputs. (b) Column compaction on 8(a).

(a)

b_1	b_2	b_3	b_4	b_5	b_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

(b)

b_1	b_2	b_3	b_4	b_5	b_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

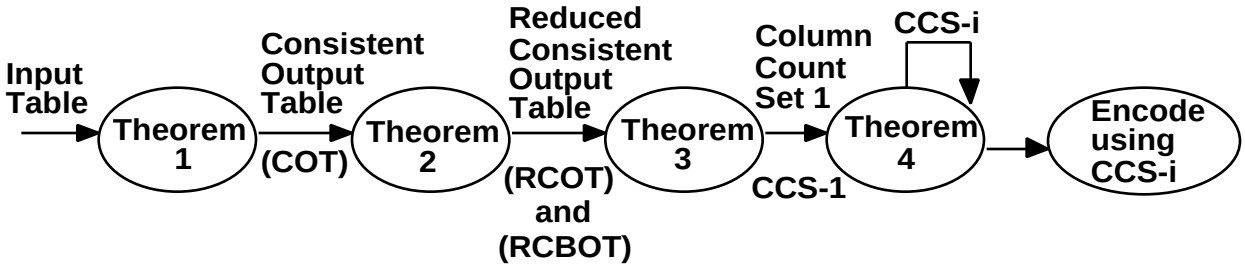

Figure 3. A High Level Flow of our output encoding algorithm (Algorithm 1).

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. Detailed Algorithm 1 is shown in Fig. 4. Theorem 5 proves the optimality of Algorithm 1.

Theorem 5: Algorithm 1 always finds the maximum *j* such that *j* encoding bits can be merged with *j* elements of the B-set, for a given encoding length.

Proof: (By Contradiction) 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 *j* 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 *(j+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 *(j+1)*. Then, *S* must satisfy CCS-*(j+1)*. But, our algorithm returned that

CCS-*(j+1)* is NULL which means that there exists no set of *(j+1)* columns that satisfies CCS-*(j+1)*. Q.E.D.

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.
m, the number of bits used to encode the output symbols

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 b_k B-set
Check if b_k is an inconsistent column
2. Form the Consistent Output Table (COT) with the consistent and the symbolic output columns.
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. Apply Theorem 3
CCS-*1* = *NULL*;
for each column b_i of RCBOT
if ((0s count in column $b_i \le 2^{m-1}$) AND (1s count in column $b_i \le 2^{m-1}$))
CCS-*1* = CCS-*1* $\{b_i\}$
5. Apply Theorem 4
j = 1
while ((*j* < *m*) AND (CCS-*j* is not *NULL*))
CCS-*(j+1)* = *NULL*;
for each X_k CCS-*j*
for each b_l CCS-*1*
candidate = X_k $\{b_l\}$
if |*candidate*| = *j+1*
Apply *(j+1)*-Column Counting Check on candidate
If the check is satisfied, CCS-*(j+1)* = CCS-*(j + 1)* *candidate*
j = *j+1*
6. Suppose that the loop of Step 5 terminated at *j* = *k*.
Choose *A* CCS-*k*
For each row *l* of the RCOT
Let *B* be the entry in the S-column and row *l*
First *k* bits in encoding of *B* = entries in row *l* and columns *A*
7. 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.
8. end

Figure 4. Pseudo-code of Algorithm 1.

A rough estimate of the time complexity of Algorithm 1 is described next. The complexity of step 1 is $rlog_2r + rc$, 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 4 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

$\binom{k}{i} i2^i n$. Thus, the net complexity of steps 4 and 5 is: $\sum_{i=1}^{m} \binom{k}{i} i2^i n$, where m is the number of bits used to encode the symbolic outputs. Steps 6 and 7 have a worst case complexity of O($nlog_2n + n$).

For the example of Table 5, to determine the third bit in the encoding of the symbolic outputs, a conventional output encoding algorithm (described in [8]) could be used. For our example, the first two bits in the encoding of Z_1 and Z_3 are equal (Table 8(a)). Hence, for determining the third bit, we specify a new output encoding problem using two symbolic outputs s_1 and s_2 so that these two outputs are distinguished. A similar case holds for Z_4 and Z_6.

Table 9. Symbolic outputs for further minimization.

b_1	b_2	b_3	b_4	b_5	b_6	New Symbol	Original Symbol (first 2 bits)
0	0	1	1	1	0	s_1	Z_1 (01)
1	0	0	1	0	1	s_1	Z_2 (00)
1	0	0	1	1	1	s_2	Z_3 (01)
0	0	0	1	0	1	s_1	Z_2 (00)
0	1	1	1	0	0	s_1	Z_4 (10)
0	1	0	1	1	0	—	Z_5 (11)
1	0	1	1	1	1	s_2	Z_3 (01)
1	1	0	0	0	0	s_2	Z_6 (10)
1	0	1	1	1	1	s_2	Z_3 (01)
1	0	0	1	0	1	s_1	Z_2 (00)

In Table 9, we distinguish between Z_1 and Z_3 by assigning s_1 to the rows corresponding to Z_1 and s_2 to the rows corresponding to Z_3. Similarly, s_1 has been assigned to the rows corresponding to Z_4 and s_2 to the rows corresponding to Z_6. In order to ensure that Z_2 has a single code assigned to it, we have assigned s_1 to each row corresponding to Z_2. Since, there is a single entry in the table corresponding to Z_5, we have left the entry corresponding to Z_5 as don't care in the above table. Further improvements are possible.

(a) If we assign 00- to Z_2, then we could allow don't care entries in the rows of Table 9, corresponding to Z_2. This allows for additional optimization. However, in this case Z_2 may have multiple codes associated with it. Although in the problem definition we mentioned that we want to assign a distinct code to each symbolic output, there are some situations where multiple codes may be allowed.

(b) Note that in Table 9, we could also serve our purpose by assigning s_2 to the rows corresponding to Z_4 and s_1 to the rows corresponding to Z_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 paper.

For Table 5, if we used 4 bits to encode the symbolic outputs instead of 3, all the outputs corresponding to the encoding bits can be merged with the elements of the reduced B-set. In that case, Z_1 will be encoded as 0110, Z_2 as 0101, Z_3 as 0111, Z_4 as 1100, Z_5 as 1110 and Z_6 as 1000. Algorithm 1 can be extended to take care of this case, and is shown as Algorithm 1.A in Fig. 5. In Algorithm 1.A, we start from the minimum number of bits required to encode the symbolic outputs and continue upto the minimum of the number of symbolic outputs and the cardinality of the reduced B-set. During each iteration, we calculate the maximum *j* such that CCS-*j* is not NULL. We break from the loop if all the encoding bits can be merged with the existing columns or we reach the upper limit of the iteration variable. Finally, we choose the encoding which generates the minimum number of extra output bit columns. The worst-case complexity of Algorithm 1.A is the worst-case complexity of Algorithm 1 multiplied by the number of symbolic outputs. Theorem 6 tells us that if the reduced B-set is sufficiently large and no two rows of the reduced B-set are identical, then there always exists an encoding of the symbolic outputs, such that all the encoding bit columns can be merged with the existing columns.

```
ALGORITHM 1.A

Steps :

1. Apply Steps 1 through 3 of Algorithm 1.
2. for (i = ⌈log2(symbolic output count)⌉; i ≤ min(symbolic output count, |reduced B-set|); i = i + 1) {
3.    Apply steps 4 and 5 of Algorithm 1 using m = i.
4.    Find the largest j, CCS-j ≠ NULL.
5.    extra_i = i - j.
6.    If (j equals i)
7.       break;  /*** We can encode using i bits so that all the outputs corresponding to the
                          encoding bits can be merged with the existing outputs ***/
   }
8. Choose the number of bits (p) for encoding the symbolic outputs such that extra_p is minimum.
```

Figure 5. Pseudo-Code of Algorithm 1.A.

Theorem 6: If there is no restriction on the number of encoding bits then, there always exists an encoding of the symbolic outputs for which all the encoding bits can be merged with the bound output columns if and only if no two rows of the RCBOT are identical.

Proof: (Sufficiency) (By Construction) Since all the rows of the RCBOT are distinct, we can construct an encoding out of *p* bits, where *p* is the number of columns in the RCBOT, in the following way. If row *i* of the RCBOT corresponds to the symbolic output s_i, then the encoding of s_i is the string of 0s and 1s in row *i* of the RCBOT.

(Necessity) Each row of the RCBOT corresponds to a distinct symbolic output. If two rows of the RCBOT are identical and still *all* the encoding bits can be merged with the already existent output columns, then two distinct symbolic outputs will be assigned the same binary code which is not allowed by the problem definition. Q.E.D.

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 described in Sec. 3. However, we extend 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 b_i B-set has specified but different values (i.e., b_i does not have a *don't care* in row *p* or *q*), then b_i cannot contribute to reducing the cardinality of the output column cover after encoding the S-column.

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).

Table 10. The Output Portion of a Specification Table.

b_1	b_2	b_3	b_4	b_5	b_6	Symbolic
0	—	1	0	—	0	Z_1
1	0	—	—	—	—	Z_2
1	—	—	—	1	—	Z_3
0	—	—	—	—	1	Z_2
0	1	0	0	0	0	Z_4
0	1	1	0	—	—	Z_5
—	—	1	1	—	—	Z_3
1	1	1	0	0	0	Z_6
—	—	—	1	1	—	Z_3
—	—	1	0	—	—	Z_2

Consider the output portion of an example specification, shown in Table 10. Here, the B-set is $\{b_1, b_2, b_3, b_4, b_5, b_6\}$. Applying Theorem 1-DC to Table 10, we find that the output column b_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 Z_2. However, the other output columns satisfy the consistency constraint mentioned in Theorem 1-DC. The consistent output table (COT) is shown in Table 11. Next, we apply Theorem 2-DC to obtain the reduced consistent output table (RCOT) from the COT.

Table 11. The Consistent Output Table for the example in Table 10.

b_2	b_3	b_4	b_5	b_6	Symbolic
—	1	0	—	0	Z_1
0	—	—	—	—	Z_2
—	—	—	1	—	Z_3
—	—	—	—	1	Z_2
1	0	0	0	0	Z_4
1	1	0	—	—	Z_5
—	1	1	—	—	Z_3
1	1	0	0	0	Z_6
—	—	1	1	—	Z_3
—	1	0	—	—	Z_2

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.

Table 12. The RCOT and RCBOT for the COT of Table 11. (a) The RCOT (b) The RCBOT

(a)

b_2	b_3	b_4	b_5	b_6	Symbolic
—	1	0	—	0	Z_1
0	1	0	—	1	Z_2
—	1	1	1	—	Z_3
1	0	0	0	0	Z_4
1	1	0	—	—	Z_5
1	1	0	0	0	Z_6

(b)

b_2	b_3	b_4	b_5	b_6
—	1	0	—	0
0	1	0	—	1
—	1	1	1	—
1	0	0	0	0
1	1	0	—	—
1	1	0	0	0

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. We apply the reduction procedure in Theorem 2-DC to the Consistent Output Table in Table 11 to obtain the RCOT and the RCBOT shown in Tables 12(a) and 12(b), respectively.

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.

We apply the Single Column Counting Check, as per Theorem 3, to the columns of the RCBOT of Table 12(b). Since the number of distinct symbols (number of rows of RCBOT) is six, *m*, the minimum number of bits required to encode the symbols, is 3. We find that column b_3 has more than four 1s while column b_4 has more than four 0s. Hence, the CCS-*1* set is $\{\{b_2\}, \{b_5\}, \{b_6\}\}$.

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 the RCBOT having any of the 2^i possible binary combinations formed by the entries under those columns in each row of the RCBOT which are fully specified (i.e., there is no *don't care* in that row under those *i* columns) and 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 the RCBOT).

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

For the RCBOT of Table 12(b), columns b_2 and b_5 satisfy CCS-*1*, when $m = 3$. For CCS-*2*, if we consider columns b_2 and b_5, we have to consider only the string 10 in the fourth and the sixth rows. This is because, entries in the other rows in these two columns (e.g. 0- in second row, -1 in the third row, etc.) contain don't cares. We find that $\{b_2, b_5\}$ satisfies CCS-*2*. Thus, while checking for CCS-*i*, we consider only 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.

We apply 2-Column Counting Check to the RCBOT of Table 12(b) according to Theorem 4-DC to the possible candidates of CCS-*2* viz. $\{b_2, b_5\}$, $\{b_2, b_6\}$ and $\{b_5, b_6\}$. We find that all of them satisfy the counting check. Finally, we try the 3-Column Counting Check (Theorem 4-DC, $i = 3$) on $\{b_2, b_5, b_6\}$. We find that *100* appears twice in row 4 and row 6 of Table 12(b) and hence violates Theorem 4-DC for $i = 3$. Hence, CCS-*3* is a null set

Let us suppose that the computation of the CCS's terminates with CCS-*j*. For the incompletely specified case, we select any member of CCS-*j*. For our example, we arbitrarily choose $\{b_2, b_5\}$, a member of CCS-*2*. Now, 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 [5].

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 $v \in V_1$. For our purpose, each member of V_1 represents a symbol and has a label which is the binary string made up of the entries the selected *j* columns in the row corresponding to that symbol in the S-column of the 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 [5][11]. 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 while 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 [5]. After solving the maximum flow problem, let us consider the set M of edges between V_1 and V_2 in the graph with non-zero flow. For any edge $(u, v) \in M$, $u \in V_1$ 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.

For our example (RCBOT of Table 12(b)), 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 b_2 and b_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 Z_1, Z_2, Z_3, Z_4, Z_5 and Z_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. 6.

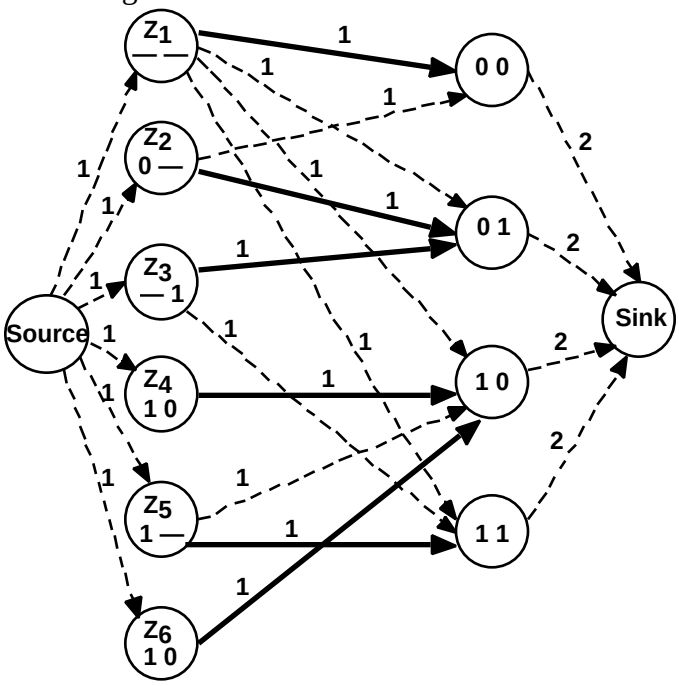

Figure 6. The Graph for the Maximum Flow Problem.

We solve the maximum network flow problem on the graph of Fig. 6, and the solid edges show the final mapping. Thus, the first two bits in the encoding of Z_1, Z_2, Z_3, Z_4, Z_5 and Z_6 are 00, 01, 01, 10, 11 and 10, respectively. Hence, the encoding of Z_1, Z_2, Z_3, Z_4, Z_5 and Z_6 are 001, 010, 011, 100, 111 and 101, respectively. Note that there is more than one possible mapping — hence, there are different possible 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 [8] and [18]. Table 13(a) shows the table obtained after encoding the symbols of Table 10. Table 13(b) is obtained after column compaction to Table 13(a).

Table 13. Output encoding. (a) Table 10 with encoded outputs (b) Column compaction on 13(a).

(a)

b_1	b_2	b_3	b_4	b_5	b_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

(b)

b_1	b_2	b_3	b_4	b_5	b_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

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*.
m, the number of bits used to encode the output symbols

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 COT consisting of the *consistent* columns and the S-Column
3. Apply Theorem 2-DC similar to Algorithm 1
4. Apply Theorem 3 similar to Algorithm 1
5. Apply Theorem 4-DC similar to Step 6 of Algorithm 1
6. Suppose that the loop of Step 5 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*.
7. Solve the maximum flow problem for *G* using Ford-Fulkerson's method.
8. 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*.
9. 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.
10. end

Figure 7. Pseudo-code of Algorithm 2.

Algorithm 2 is described in details in Fig. 7. The proof of the optimality of Algorithm 2 is similar to the optimality proof of Algorithm 1 (Theorem 5). The complexity of Algorithm 2 is that of Algorithm 1 plus the complexity of the matching problem solved using the Ford-Fulkerson method. As mentioned in [5], the complexity of the Ford-Fulkerson method, implemented using Edmonds-Karp's algorithm, has a

complexity of $O(VE^2)$, where V and E are the number of vertices and edges, respectively, in the graph under consideration. For our case, the number of vertices in the graph is $V = n + 2^k + 2$, where n is the number of symbolic outputs and k is the number of encoding columns that can be merged with the already existent output columns. Note that, the *2* in the expression is due to the presence of source and sink nodes. The number of edges in a graph is $E = O(V^2)$. So, the *net* complexity of this step is $O(V^5)$. However, if we calculate the actual number of edges, the count can be upper bounded by $n + 2^k + n2^k$. This bound comes from the fact that the source node is connected to n vertices and 2^k vertices are connected to the sink node. Now, each element of V_1, representing each symbolic output, can be connected to atmost 2^k elements of V_2 (if its label consists of all *don't cares*).

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 or 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* [19]. 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 14. 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 [16]. For both cases, we used NOVA [21] 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 [13] for technology mapping. Table 14 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.

In Table 14, we have also reported the number of extra output columns that were generated by our technique because all the encoding bits could not be merged with the already existent output columns. The time taken to run our algorithms was extremely

small on a Sun Ultra-SPARC-2 machine for all these examples — hence, we did not report the individual runtimes.

Table 14. Experimental results : Area comparisons using our output encoding algorithm (best case) withthe output encoding algorithm which never merges any output columns (Worst Case).

MCNC FSM Name	3 bits for encoding				4 bits for encoding			
	Our Algorithm		Worst Case		Our Algorithm		Worst Case	
	Area	Extra Bits	Area	Extra Bits	Area	Extra Bits	Area	Extra Bits
bbara	*205	1	225	3	*285	2	346	4
bbsse	316	0	364	3	316	0	378	4
bbtas	*98	1	99	3	*103	2	147	4
dk14	223	0	266	3	223	0	294	4
dk15	206	0	208	3	206	0	257	4
dk17	*182	0	191	3	*231	1	304	4
ex1	721	0	750	3	721	0	781	4
ex6	254	0	312	3	254	0	405	4
mc	93	0	96	3	93	0	96	4
opus	235	0	260	3	235	0	356	4
tav	73	0	143	3	73	0	188	4
tma	444	0	543	3	444	0	548	4

* : 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 BASIC GATES

Let us consider the output part of a very simple specification, shown in Table 15. It has a single symbolic output column and two elements in the B-Set, b_1 and b_2. There are two output symbols, s_1 and s_2, and hence, a single bit is sufficient to encode the symbols. Neither b_1 nor b_2 is consistent because b_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 b_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 b_1 and b_2, then we find that $AND_{1,2}$ is a consistent column and can be used to encode s_1 and s_2. In that case, we can encode s_1 as 0 and s_2 as 1. This leads to an extension of our output encoding technique, described in Sec. 3 and 4, in the following way.

Table 15. Extension of the output encoding technique using basic gates.

b_1	b_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

Let us consider the output part of a very simple specification, shown in Table 15. It has a single symbolic output column and two elements in the B-Set, b_1 and b_2. There

are two output symbols, s_1 and s_2, and hence, a single bit is sufficient to encode the symbols. Neither b_1 nor b_2 is consistent because b_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 b_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 b_1 and b_2, then we find that $AND_{1,2}$ is a consistent column and can be used to encode s_1 and s_2. In that case, we can encode s_1 as 0 and s_2 as 1. This leads to an extension of our output encoding technique, described in Sec. 3 and 4, in the following way.

Let us define *G* as a set of *basic* gates. *G* may consist of two or three input AND and OR gates. Note that the inverter need not be considered in *G* because it has already been considered implicitly in Sec. 3 and 4. Similar reasoning holds for NAND and NOR gates. For the purpose of a simple example, let us consider that *G* 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 b_i and b_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 $AND_{i,j}$ is a *don't care*. In general, if one of the entries in column b_i or b_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 b_i and b_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 (members of the B-set), form the *Extended Bound Set* (EB-Set).

The general framework that we developed in Sec. 3 and 4 can now be used to solve the output encoding problem with the EB-set under consideration instead of the B-set. However, we have an additional step. After calculating the CCS-*i* sets, our algorithm (Step 6 of Algorithm 1) chose any element of CCS-*j* (where *j* is the largest integer such that CCS-*j* is not empty) for performing the actual encoding; instead, now we choose the element of the CCS-*j* set which has the minimum number of derived columns. It may be noted that the extension of our technique to include basic gates has a drawback that the size of the initial set of columns increases — however, this approach is useful for the cases where the cardinality of the initial reduced B-set is small.

7. CONCLUSION

In this paper, we have presented a new output encoding algorithm whose objective is to encode the symbols in the symbolic output column of a specification table in such a way that the number of output functions, after performing the encoding and subsequent output column compaction, is minimum. The algorithm has many practical applications. It can be used as a pre-processing step during conventional output or FSM state encoding. Another important application of this algorithm is to generate logic circuits with low-area overhead that always maintain one-hot encoding on certain signals, even during scan or BIST operations, as discussed in [16]. 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. 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) even for large benchmark circuits. An interesting extension of this algorithm will be to consider specifications with bound output columns (outputs specified using 1s, 0s and don't cares) and multiple symbolic output columns.

8. ACKNOWLEDGMENTS

The authors wish to thank Dr. Nirmal Saxena, Dr. Jonathan Chang and Philip Shirvani of Stanford Center for Reliable Computing 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

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

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

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

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] 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.

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

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

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

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.