An Output Encoding Problem and a Solution Technique

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

Center for Reliable Computing
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 1's, 0's and don't cares, and others are specified symbolically, and assuming that the minimum number of bits are used to encode the symbolic outputs ($\lceil \log_2 n \rceil$ bits for n symbolic outputs), 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 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. We develop an exact algorithm to solve the above problem and present experimental data to validate the claim that our encoding strategy helps to reduce the area of a synthesized circuit.

1. INTRODUCTION

Column compaction plays a major role in the synthesis of digital systems [1]. In *column compaction*, the number of output functions for a given specification is reduced by merging outputs which are logically equivalent, or can be made equivalent through assignment of *don't cares*. Two examples of output column compaction are shown in Fig. 1. Given a set of output columns, the problem of finding the smallest set that can be obtained by compacting the given set is related to the *maximum clique partitioning problem*, which is an NP-Complete problem [2]. This smallest set of compacted outputs is called the *minimum cardinality output column cover*. Column compaction can greatly reduce circuit area; this is not only true for the PLA implementation of the circuit [3] but also for multi-level logic circuits.

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

Figure 1. Examples of column compaction.

The different types of encoding problems, studied in the past, are the *finite state machine (FSM) state encoding* problem and the *input* and the *output* encoding problem. Techniques for FSM state encoding have been discussed in [4, 5, 6, 7, 8, 9, 10]. The input encoding problem has been reported in [11, 12]. 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 [13]. Another version of the output encoding problem is discussed in [14]. 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 [15]. However, this algorithm does not consider any existing non-symbolic (binary) outputs when determining the encoding for the symbols of the symbolic output column.

In this paper, we present an output encoding problem whose objective is different from those mentioned above. 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 0's, 1's and *don't cares* while the other outputs are symbolically specified. The problem is to encode the symbols in the symbolic output column, using $\lceil \log_2 n \rceil$ bits for n symbolic outputs, so that after encoding, the cardinality of the output column cover computed using column compaction is minimum. 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 of Sec. 3 to handle *don't cares* in the outputs of the specification. Experimental results are reported in Sec. 5 followed by conclusion in Sec. 6.

2. MOTIVATION

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 1's, 0's 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. In this paper, we will consider specification tables containing a single symbolic output column (S-column) for simplicity. We can handle cases with multiple symbolic output columns by choosing an appropriate ordering of the symbolic outputs and repeatedly applying the algorithm reported in this paper.

For Table 1, since we have seven distinct symbols in the S-column, we need 3 bits to encode them. 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 after column compaction. In Table 2(b), c_5, c_6 and c_7 represent the symbolic outputs.

Table 1 . Truth table with symbolic output.

Input	Encoded Output 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). Encoding.

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). Compacted Truth Table.

Input	Outputs						
	c_1	c_2	c_3	c_4	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

Table 3(a). Encoding.

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

3(b). Compacted Truth Table.

Input	Output $\underline{c_1}$ $\underline{c_2}$ c_3 $\underline{c_4}$
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

It is possible to reduce the number of output columns after column compaction if we encode the symbols of Table 1 as shown in Table 3(a). The column compacted truth table is shown in Table 3(b). In Table 3(b), columns c_1, c_2 and c_4 represent the symbolic outputs.

This output encoding problem has many applications. Digital systems are often specified in terms of a combination of binary valued and symbolic signals. This problem is applicable to reduce the area of scan-based [16] designs containing one-out-of-n (one-hot) signals, as described in [17]. This algorithm can serve as a pre-processing step for FSM state encoding.

3. ENCODING ALGORITHM FOR FULLY SPECIFIED OUTPUTS

In this section, we present our output encoding algorithm with the assumption that the output columns belonging to the B-set are fully specified with 1's and 0's for simplicity. We consider the example of Table 4 for illustration. We describe the three basic steps of our algorithm as follows:

Step 1: Find Inconsistent Columns: Find all pairs of rows, *p* and *q*, in Table 4 for which the S-column has the same value and $c_i \in$ B-set has different values. Then c_i is said to be *inconsistent*. For this example, the B-set is $\{c_1, c_2, c_3, c_4, c_5, c_6\}$. Here, c_1 has a 1 in the second row and a 0 in the 4th row; but the S-column has X_2 in both of these rows. Hence, c_1 is inconsistent. Column c_3 is also inconsistent for symbolic output X_3.

Step 2: Create reduced consistent output table: Remove inconsistent columns from the specification table to obtain the *consistent output table* (COT). Merge equal rows of COT to obtain *reduced consistent output table* (RCOT). Table 5 is the RCOT for Table 4. Since there are 6 symbolic outputs, we use $m = 3$ bits to encode them.

Step 3: *i*-column counting check: Any set of *i* columns of the RCOT ($i \leq m$) having any of the 2^i possible binary values of length *i* appearing more than 2^{m-i} times in the rows of the RCOT cannot reduce the size of the output column cover after encoding the symbols. All sets of *i* columns that satisfy this *i*-column counting check form elements of *column-count-set-i* (CCS-*i*). For Table 5, CCS-1 = $\{\{c_2\}, \{c_5\}, \{c_6\}\}$, CCS-2 = $\{\{c_2, c_5\}, \{c_5, c_6\}, \{c_2, c_6\}\}$, CCS-3 = NULL. $\{c_4\}$ is not included in CCS-1 because the number of 1's in the c_4 column is 5 (greater than 4). We terminate this step when either a CCS set is NULL or CCS-*m* has been generated.

Table 4 . Specification Table.

c_1	c_2	c_3	c_4	c_5	c_6	Symbol
0	0	1	1	1	0	X_1
1	0	0	1	0	1	$\mathbf{X_2}$
1	0	**0**	1	1	1	$\mathbf{X_3}$
0	0	0	1	0	1	$\mathbf{X_2}$
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	$\mathbf{X_3}$
1	0	0	1	0	1	X_2

Table 5 . The RCOT.

c_2	c_4	c_5	c_6	Symbol
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

For Table 4, we can encode the symbols using three bits in such a way that two of the three columns, representing the symbolic outputs, can be merged with the existing output columns by column compaction. 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. We determine the third bit such that the codes assigned to the symbols are distinct. Thus, referring to Table 4, X_1 can 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. It is straightforward to prove the optimality of our solution [18]. It should be noted that if we allowed 4 bits to encode the symbolic outputs of Table 4, then all output columns corresponding to the four encoding bits could be merged with c_2, c_4, c_5 and c_6. In that case, we can iterate step 3 incrementing the number of encoding bits until all the encoding bit outputs can be merged with the existing outputs or we reach a point where we have tried to encode *n* symbolic outputs using *n* bits.

4 . ENCODING ALGORITHM FOR INCOMPLETELY SPECIFIED OUTPUTS

The basic steps of the algorithm described in Sec. 3 can be extended to handle don't cares in the B-set.

Table 6 . Specification Table.

c_1 c_2 c_3 c_4 c_5 c_6	Symbolic
0 - 1 0 - 0	X_1
1 0 - - - -	$\mathbf{X_2}$
1 - - - 1 -	X_3
0 - - - - 1	$\mathbf{X_2}$
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 7 . The RCOT.

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
0 1 0 0 0	X_4
1 1 0 - -	X_5
1 1 0 0 0	X_6

Consider the specification shown in Table 6. The RCOT is shown in Table 7. The CCS-1 set is $\{\{c_2\}, \{c_5\}, \{c_6\}\}$. For calculating CCS-2 and CCS-3 we must ensure that the count of the *fully specified* binary strings (binary strings containing no don't cares) only need to satisfy the bounds discussed in step 3 of the algorithm in Sec. 3. Thus, CCS-2 is $\{\{c_2, c_5\}, \{c_2, c_6\}$ and $\{c_5, c_6\}\}$. 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 solve the problem of assigning 0's and 1's to the *don't cares* in columns c_2 and c_5 so that the output columns representing the symbolic outputs can be merged with c_2 and c_5. We formulate the problem as a bipartite graph matching problem [2]. We form a weighted bipartite graph as follows: Set V_1 contains vertices corresponding to each symbolic output with a label equal to the string formed by the entries in the row corresponding to that symbolic output under columns c_2 and c_5 of Table 7. Set V_2 contains 4 vertices each having a label that is a distinct binary pattern of length 2. For each $u \in V_1$ and $v \in V_2$ we have an edge (u, v) of weight 1 if and only if the label of v contains 1(0) in all positions in which the label of u contains 1(0). We add two more vertices to the graph — the *source* and the *sink*. There is an edge of weight 1 from the source vertex to each member of V_1. There is an edge of weight $2^{3-2} = 2$ from each member of V_2 to the sink.

The graph along with the source and the sink nodes and the edge weights is shown in Fig. 2. We solve the maximum network flow problem on the graph using the Ford-Fulkerson method [19]. The solid edges show the final mapping. 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, one encoding of X_1, X_2, X_3, X_4, X_5 and X_6 is 001, 010, 011, 100, 111 and 101, respectively. Note that there may be multiple solutions to this bipartite graph matching problem. Hence, our algorithm can be further refined by selecting a mapping of members of V_1 to the members of V_2 based on some heuristic following the observations developed in [13] and [14].

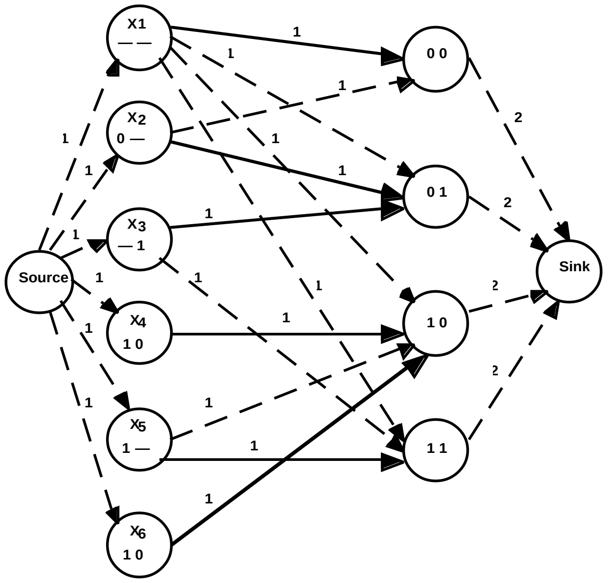

Figure 2 . The Graph for the Maximum Flow Problem.

5. EXPERIMENTAL RESULTS

In this section, we present 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 either 3 or 4.

Table 8 . Area results for our output encoding algorithm (best case) and the worst case which never merges any output column.

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 by our algorithm because the cardinality of the B-set is less than the number of bits needed to encode the symbols.

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 existing outputs (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* [20]. 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 8. 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 [17]. For both cases, we used NOVA [10] 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 [21] for technology mapping. Table 8 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 already existent non-symbolic output columns by compaction.

6. 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 the specification table using $\lceil \log_2 n \rceil$ bits for n symbolic outputs 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 [17]. This algorithm can also be used as a pre-processing step for FSM state encoding. 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. As the experimental results show, 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) than worst-case circuits generated without considering output column compaction during the encoding of the symbols. In this paper, we have considered specifications with a single symbolic output column. In case of multiple symbolic output columns, we can integrate our technique with the output encoding algorithm reported in [15] to achieve a minimum number of output columns in the final table.

7. ACKNOWLEDGMENT

This work was supported by the Advanced Research Projects Agency under prime contract No. DABT63-94-C-0045. Thanks to Robert B. Norwood and Jonathan T. Y. Chang of Center for Reliable Computing, Stanford University, for their comments regarding this work.

8. REFERENCES

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

[2] Cormen, T. H., et. al., *Introduction to Algorithms*, The MIT Press and McGraw-Hill, 1989.

[3] Brayton, R. K., et. al., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Publishers, 1984.

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

[5] 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.

[6] De Micheli, G., et. al., "Optimal State Assignment for Finite State Machines," *IEEE Trans. on CAD.*, CAD 4(3), 269-285, July 1985.

[7] Du, X., et. al. "MUSE: A MUltilevel Symbolic Encoding Algorithm for State Assignment," *IEEE Trans. on CAD*, 10(1), pp. 28-38, Jan. 1991.

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

[9] 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.

[10] 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.

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

[12] Yang, S., et. al., "Optimum and Sub-optimum Algorithms for Input Encoding and Its Relation to Logic Minimization," *IEEE Trans. on CAD*, 10(1), pp. 4-12, Jan. 1991.

[13] 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.

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

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

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

[17] Mitra, S., L. J. Avra and E. J. McCluskey, "Scan Synthesis for One-hot Signals", *Proc. ITC*, Nov. 1997.

[18] Mitra, S., L. J. Avra and E. J. McCluskey, "An Output Encoding Problem and a Solution Technique", *Technical Report, Center for Reliable Computing, Stanford University*, CRC TR 97-1, 1997.

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

[20] Sentovich, E., *et. al.*, "SIS: A System for Sequential Circuit Synthesis", *ERL Memo. No. UCB/ERL M92/41*, Department of EECS, UC Berkeley, CA 94720.

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