VLSI Design
1994, Vol. 2, No. 3, pp. 209–221
Reprints available directly from the publisher
Photocopying permitted by license only

Techniques for Self-Checking Combinational Logic Synthesis

FADI BUSABA and PARAG K. LALA
Department of Electrical Engineering, North Carolina A&T State University, Greensboro, NC 27411

(Received October 12, 1992; Revised March 15, 1993)

This paper presents techniques for designing arbitrary combinational circuits so that any single stuck-at fault will result in either single bit error or unidirectional multibit error at the output. If the outputs are encoded using Berger code or m-out-of-n code, then the proposed technique will enable on-line detection of faults in the circuit. An algorithm for indicating whether a certain fault at an input will create bidirectional error at the output is presented. An input encoding algorithm and an output encoding algorithm that ensure that every fault will either produce single bit error or unidirectional multibit error at the output are proposed. If there are no input fault which produces bidirectional error, no internal stuck-at fault will result in such an error irrespective of the way the circuit is implemented. Thus, only single bit or unidirectional multibit error will result in the presence of a fault in the circuit. The proposed techniques have been applied to MCNC benchmark circuits and the overhead is estimated.

Key Words: *Error-detection, Self-checking, Logic synthesis, Unidirectional errors, Input encoding, Output encoding*

1. INTRODUCTION

With the increase in the complexity and density of VLSI chips, transient/intermittent faults have emerged as the dominant failure modes in VLSI circuits [1–2]. Conventional off-line testing schemes do not detect transient faults since the detection of these faults requires continuous monitoring of the outputs; i.e. the circuits have to be self-checking. A self-checking circuit usually consists of a functional block that generates the encoded outputs, and a checker that checks the validity of the outputs [3–5]. Self-checking circuits that use m-out-of-n code [6] or Berger code [7] for output encoding, detect stuck-at faults that cause single bit error or unidirectional multibit error. Designing logic circuits such that any stuck at-fault causes single bit error or unidirectional multibit error is a challenging problem. Previous work either use PLA structure, or perform algebraic factorization for two-level networks by restricting the use of inverters at the inputs and using only AND/OR gates [8–9]. Also, it is assumed that all input lines and their inversions are fault free. In [14], monotone functions and inverter free realization were used to design strongly fault secure logic networks. Similarly, inverter free realizations were used in [15] for the design of strongly fault secure and strongly code disjoint circuits. The restriction on the type of gates and on synthesis procedures used for logic circuits usually increases the area overhead. In this paper, we propose techniques for designing bidirectional error-free combinational circuits based on input encoding and output encoding schemes that do not restrict the way the circuit is implemented. The output encoding strategy does not form codewords as used in conventional coding techniques. Our intention here is to design the functional part of a self-checking circuit such that any single stuck-at fault will create either single bit error or unidirectional multibit error. Thus by incorporating additional check bits, the output of the functional block can be designed to be a single and unidirectional error detecting code; e.g., Berger code and m-out-of-n code.

The techniques presented in this paper can be directly applied to logic circuits described in the PLA format, where symbolic representation is used for the inputs or the outputs or both; i.e., the inputs or the outputs have not been assigned any binary codes. The aim of this paper is to encode the inputs or the

outputs so that any single stuck-at fault can only create single bit error or unidirectional multibit error at the output. Thus, these techniques will be useful only if self-checking is considered at the design level i.e., before the circuit is actually implemented.

Preliminaries and definition are given in section 2. In sections 3, an algorithm for detection of faults that might cause bidirectional error at the outputs is presented. An input encoding algorithm and an output encoding algorithm are presented in section 4. A detailed example is given at the end of section 4 where the input and output encoding algorithm are applied to a benchmark circuit. A VLSI implementation of each technique is provided, and the area overhead is estimated. The proposed techniques have been applied to MCNC benchmark circuits, and the results are reported in section 5.

2. PRELIMINARIES

Before presenting the proposed algorithms, we need to consider the following definitions.

Definition 1: A **variable** is a symbol representing a single coordinate of the boolean space (e.g. a).

Definition 2: A **literal** is a variable or its negation (e.g. a or a').

Definition 3: A **cube** is a set C of literals such that if x belongs to C, then x' does not belong to C.

Definition 4: A **minterm** is a cube with only 0 and 1 entries.

Definition 5: A function is **unate** if in its minimal sum-of-products expression each variable appears either in a complemented form or in an uncomplemented form but not in both.

Definition 6: The **dist**(C_1, C_2), where C_1 and C_2 are different input cubes, is the number of bit positions in which they differ when these bit positions are either 1's or a 0's, but not don't cares. For example dist(C_1, C_2), where C_1(abcd) = 1010 and C_2(abcd) = -110, equals to one because the two input cubes differs only in position b.

Definition 7: Two output vectors O_1 and O_2 are **partially bidirectional** if there exist at least two outputs bits which are 10 in O_1 and 01 in O_2 or vice versa. For example, O_1 = 1100 and O_2 = 0101 are partially bidirectional whereas O_1 = 1000 and O_2 = 1111 are not.

Definition 8: Two output vectors O_i and O_j are **adjacent** if dist(C_i, C_j) = 1, where C_i and C_j are the input cubes corresponding to O_i and O_j respectively.

Definition 9: Two input cubes C_i and C_j are **bidirectionally adjacent** if their corresponding outputs, O_i and O_j, are partially bidirectional.

Definition 10: Two input cubes are called ***m*-bidirectional** if they only differ in position m and their corresponding outputs are partially bidirectional. For example, input cubes C_1 and C_3 in Table I are a-bidirectional.

Definition 11: **Bidirectionality set** of two bidirectionally adjacent cubes is the set that contains all the variables in which the two cubes differ. For example, the bidirectionality set of C_1 and C_2 in Table I is {b, c} since they differ in positions b and c.

Definition 12: The **num_inv(n1, n2)** is the number of inversions on the path from n1 to n2 modulo 2, where n1 and n2 are two nodes inside a digital circuit.

Definition 13: A fault f creates a **unidirectional error** if the correct and the faulty outputs are not partially bidirectional.

Definition 14: An **undirected graph** G containing the set of vertices V and the set of edges E is denoted by G(V, E). G'(V', E') is a **subgraph** of G if V' is a subset of V, and an edge joins two vertices in G' if an edge joins the same two vertices in G. A **fully connected subgraph** of G is a graph $G_s(V_s, E_s)$ such that V_s is subset of V, and any two vertices in G_s are connected by an edge. A graph $G_1(V_1, E_1)$ **covers graph** $G_2(V_2, E_2)$ if the following are satisfied:

(i) the number of vertices in E_1, is equal to or greater than the number of vertices in E_2.

(ii) if there exists an edge between two nodes in V_2, there should be also an edge between the same two nodes in V_1.

Definition 15: A graph, G_m, containing 2^m vertices is **unidirectional graph** iff

(i) each vertex is uniquely represented by an m-bit number.

(ii) any two vertices are connected if they are not partially bidirectional.

The following definitions are taken from [12].

Definition 16: A **controlling value** at a gate input is the value that determines the value at the output of the gate independent of the other inputs. For example, 0 is a controlling value for an AND gate. A **noncontrolling value** at a gate input has no effect on

TABLE I
Truth Table of a Logic Circuit

	inputs				Symbolic	outputs		
cubes	a	b	c	d	outputs	O_1	O_2	O_3
C_1	0	1	0	1	Z_1	1	0	0
C_2	0	0	1	1	Z_2	0	1	0
C_3	1	—	0	1	Z_2	0	1	0

the output of the gate. For example, 1 is a noncontrolling value for an AND gate.

Definition 17: A **path** is in a combinational circuit consists of connections and gates, where *connection i* connects *gate* $(i - 1)$ and *gate i*. In other words, an input to *gate i* is the output from *gate* $(i - 1)$. The inputs to a *gate i* other than the output from *gate* $(i - 1)$ are called **side inputs.**

Definition 18: A path is said to be **statistically sensitizable** if there exists an input cube that sets all side inputs to noncontrolling values.

Definition 19: An **event** is the transition from 0(1) to 1(0).

Definition 20: A primitigve gate is **prime** is none of its inputs can be removed without causing the resulting circuit to be functionally different. A gate is **irredundant** if its removal causes the resulting circuit to be functionally different. A gate-level circuit is said to be **prime** if all the gates are prime and **irredundant** if all the gates are irredundant. A gate-level circuit is prime and irredundant if and only if it is 100% testable for all single stuck-at faults [13].

3. DETECTION OF BIDIRECTIONAL FAULTS AT INPUT LINES

A fault at a node f results in unidirectional error at the outputs if the number of inversions from the fault site to the outputs is the same; in other words, num_inv(f, o_i) is the same, where o_i is an input line that is affected by f.

The following two lemmas identify the existence of bidirectional error due to a fault on an input line, and how to eliminate such errors.

Lemma 1: If there exist two x-bidirectional input cubes, then a fault at input x may create a bidirectional error at the output.

Proof: Suppose cubes C_1 and C_2 are x-bidirectional, then C_1 and C_2 differ only in variable x and the outputs corresponding to C_1 and C_2 i.e., O_1 and O_2, are partially bidirectional. Therefore, if C_1 is activated, a fault at input x may activate C_2 instead of C_1 producing O_2 instead of O_1, thus creating a bidirectional error. Q.E.D. □

Lemma 2: If no two input cubes are x-bidirectional, than a fault at an input line will always produce a unidirectional error.

Proof: If no x-bidirectional input cubes exist, then either (1) the distance between all bidirectionally adjacent input cubes is zero or greater than or equal to two, or (2) any adjacent outputs are not partially bidirectional. For case (1), no fault at an input line will activate cube C_j instead of C_i where C_i and C_j are m-bidirectional; thus every fault results in unidirectional error at the output. For case (2), a fault at the input may activate C_j instead of C_i. Since the outputs corresponding to C_i and C_j are not partially bidirectional, the fault results in unidirectional error at the output. Q.E.D. □

TABLE II
Specification of a Full Adder

cubes	A	B	C	O_1	O_2
C_1	1	0	0	1	0
C_2	0	1	0	1	0
C_3	0	0	1	1	0
C_4	1	1	1	1	1
C_5	1	1	0	0	1
C_6	0	1	1	0	1
C_7	1	0	1	0	1
C_8	0	0	0	0	0

The following algorithm identifies which faults at the input lines might cause bidirectional error at the output.

Algorithm 1:

1. For every possible output O_i in a circuit, group all other outputs that are partially bidirectional with O_i.
2. For an output O_j in the group, find dist(C_i, C_j) where C_i is the cube corresponding to O_i, and C_j is a cube corresponding to O_j.
3. If dist(C_i, C_j) equals to 1 and the two cubes differ in variable x, then add (C_i, C_j, x) to the set of possible bidirectional faults.

To illustrate the application of the algorithm, let us consider the truth Table description of a Full Adder shown in Table II.

The first step results in the following group of outputs (10, 01). By applying step 2 and step 3, the following set is obtained: {(C_1, C_5, B), (C_1, C_7, C), (C_2, C_5, A), (C_2, C_6, C), (C_3, C_6, B), (C_3, C_7, A)}. The element (C_1, C_5, B) indicates that cubes C_1 and C_5 are B-bidirectional; i.e., if C_1 is applied to the circuit and input B gets stuck-at-1, then the input will correspond to C_5, which will result in the output 01 instead of the fault free output 10.

4. TECHNIQUES FOR BIDIRECTIONAL ERROR ELIMINATION

Preventing faults at the input lines of a circuit from causing bidirectional error at the output can be guar-

anteed by 'properly' encoding the inputs or the outputs. The input or output encoding should be done such that no two input cubes become *m*-bidirectional, thus eliminating the possibility of bidirectional error at the output. (Lemma 2). We will first describe the input encoding strategy.

4.1 Input Encoding

The input encoding strategy assigns codes to two bidirectionally adjacent inputs cubes, C_i and C_j, such that dist(C_i, C_j) is either ≥ 2, or equal to 0. We consider the two cases separately.

Case 1: In this case, C_i and C_j are coded such that dist(C_i, C_j) ≥ 2. Consider a digital circuit with I different input symbols and O encoded outputs. The steps required to satisfy the distance requirement are:

1. For each possible input cube in a circuit, group all other input cubes that are bidirectionally adjacent.
2. Assign codes with distance greater than or equal to 2 for any bidirectional adjacent input cubes by using minimum number of encoding bits.
3. If the assignment is not possible, increase the number of encoding bits by one and repeat step 2.

Consider the example shown in Table III in which each input cube is bidirectionally adjacent with four other cubes. Steps 1 and 2 suggest that cubes C_1, C_2, C_3 and C_4 have to be at a distance greater than or equal to two from cubes C_5, C_6, C_7 and C_8. This can be illustrated with a graph (Figure 1) which is constructed with input cubes as vertices, and with an edge connecting two vertices if they are bidirectionally adjacent. Thus the distance should be greater than or equal to 2 for any vertices joined by an edge. If minimum number of bits for encoding is used and

TABLE III
Description of a Logic Circuit

Input Cubes	Outputs O_1	O_2
C_1	1	0
C_2	1	0
C_3	1	0
C_4	1	0
C_5	0	1
C_6	0	1
C_7	0	1
C_8	0	1

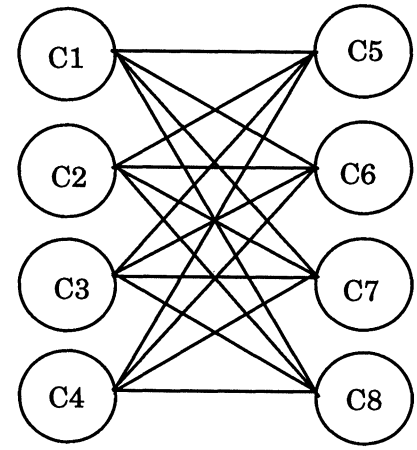

FIGURE 1

C_1 is assigned to 000, then cubes C_5, C_6, C_7 and C_8 have to be coded as 101, 110, 011 and 111 respectively. Consequently, any code for C_2, C_3 or C_4 will have a distance of one from their bidirectionally adjacent cubes which conflicts with the constraint in step 2. If four bits are used to encode the input, one possible input coding is:

$$0000 \longrightarrow C_1 \qquad 0010 \longrightarrow C_2$$

$$0100 \longrightarrow C_3 \qquad 0110 \longrightarrow C_4$$

$$1101 \longrightarrow C_5 \qquad 1111 \longrightarrow C_6$$

$$1001 \longrightarrow C_7 \qquad 1011 \longrightarrow C_8$$

If the cubes are encoded in this manner, cubes C1, C2, C3 and C4 can be reduced to the cube 0--0, and similarly cubes C5, C6, C7, and C8 can be reduced to 1--1. Therefore, any fault at the input will result in unidirectional error at the output.

An alternative approach for input encoding is to use m-out-of-n codes since the minimum distance between any two codes is two. In addition, they are easily implemented without significant increase in the number of input lines.

Case 2: In this case, C_i and C_j are coded such that dist(C_i, C_j) = 0. Let us assume two *x*-bidirectional input cubes, C_i and C_j, where the variable *x* is 0 in C_i and 1 in C_j. Encoding the input by adding one more line for the variable *x* such that it is represented by -0 in C_i and by 1- in C_j will make dist(C_i, C_j) = 0. In the proposed input encoding strategy, a 0 and a 1 in the input cubes are replaced by -0 and 1- respectively. Consider for example two *a*-bidirec-

tional input cubes C_1 (abcd) = 1010 and C_2 (abcd) = 0010. Since a is a 1 in C_1 and 0 in C_2, the new input encoding, $C_1(aa_1bcd)$ = 1-010 and $C_2(aa_1bcd)$ = -0010, will guarantee that dist(C_1, C_2) is zero, and thus the fault at input a that might cause bidirectional error at the ouput is eliminated. It is important to note that input variable a (a_1) appears either as a don't care '-' or as a 1 (0) in the input cubes. If for every input variable x, there exist at least two x-bidirectional input cubes, the number of input lines will be doubled because x will be replaced by two inputs x and x_1. Thus, in the worst case, this strategy doubles the number of input lines. In such a situation, every input variable will be either present as a 0 or 1 but not both in the input cubes.

In addition, this encoding process may result in x-bidirectional input cubes even in the absence of x-bidirectional input cubes in the original specification. For example, in Table I, C_1 and C_3 are a-bidirectional, and cubes C_1 and C_2 are bidirectionally adjacent with their corresponding bidirectionality set {b, c}. To make dist(C_1, C_3) zero, one more input variable is added such that C_1 = -0101, C_2 = -0011 and C_3 = 1--01. In this case, cubes C_1 and C_2 become c-bidirectional; that is because the bidirectionality set of C_1 and C_2 was {a, c} and by replacing a with -0 and 1- in C_1 and C_2 respectively, variable a can be taken out from the set.

The input encoding algorithm proposed below satisfies the distance requirement.

Algorithm 1:

```
Repeat
{
  For every input cube Ci
    For every input cube Cj
      If Ci and Cj are x-bidirectional
      {
        increase the number of input encoding bits
        by one as follows:
        substitute in all cubes the location of x by
        1- if it is a 1, by 0- if it is a 0 and by -- if it
        is a - (don't care).
      }
} until there is no x-bidirectional cubes.
```

After applying algorithm 1 to the circuit description of Table II, another circuit description, Table IV, is derived with different input encoding.

The new input encoding doubles the number of variables but keeps the number of literals unchanged. In this case, the output equations are:

$$O^1 = A\overline{B}^1\overline{C}^1 + \overline{A}^1B\overline{C}^1 + \overline{A}^1\overline{B}^1C$$

$$O_2 = AB + BC + AC$$

TABLE IV
Specification FA After Input Encoding

cubes	A	A_1	B	B_1	C	C_1	O_1	O_2
C_1	1	—	—	0	—	0	1	0
C_2	—	0	1	—	—	0	1	0
C_3	—	0	—	0	1	—	1	0
C_4	1	—	1	—	1	—	1	1
C_5	1	—	1	—	—	0	0	1
C_6	—	0	1	—	1	—	0	1
C_7	1	—	—	0	1	—	0	1
C_8	—	0	—	0	—	0	0	0

These output functions are unate since the variables either appear in their complemented or uncomplemented form but not both.

Lemma 3: If the minimized output functions are unate, every fault (internal or at the input lines) will cause either single bit error or unidirectional multibit error at the output regardless of the way the circuit is implemented.

Proof: If the outputs functions are unate, variables appear in either complemented or non-complemented form but not both. Thus, if a variable appears in a non-complemented form in a function, the variable either does not get inverted or go through even number of inversions from the input to the output. Similarly, if a variable appears in complemented form, it either does not get inverted or goes through odd number of inversions. Therefore, if a variable is in uncomplemented (complemented) form, and a fault occurs at the corresponding input line, this fault will create unidirectional error since there is even (odd) number of inversions from the input line to the outputs. On the other hand, if a fault exists at node x and affects k outputs, then, there will at least k different paths P1 to Pk from the fault site to the k affected outputs. Suppose node x is affected by one of the inputs, say a (Figure 2). Since the number of inversions (modulo 2) between input a and the outputs is the same,

num_inv(a, o_1) = num_inv(a, o_2) = .. = num_inv(a, o_k). In other words,

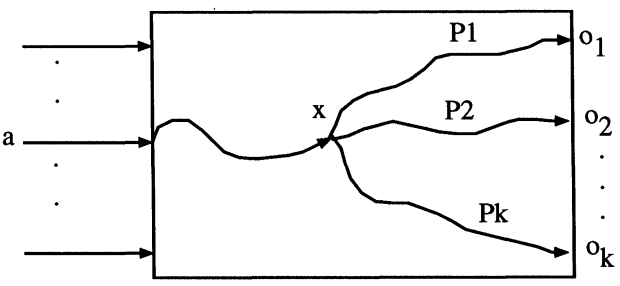

FIGURE 2. Logic Circuit Representation.

num_inv(a, x) + num_inv(x, o_1) = .. = num_inv(a, x) + num_inv(x, o_k). This implies

num_inv(x, o_1) = num_inv(x, o_2) = .. = num_inv(x, o_k)

Therefore, there is same number of inversions between the fault site and the outputs which makes the fault produce unidirectional error. Q.E.D. □

Corollary 1: If an input variable, *a*, is present in complemented or un-complemented form but not both in the output equations, then any fault located in a path leading from variable *a* to the outputs will cause either single bit error or unidirectional multibit error.

Proof: Suppose that variable *a* is presented in its uncomplemented form in all output equations, and suppose a fault *f* occurred at node *x*. Since *x* is affected by variable *a*, then by lemma 3, num_inv(x, o_i) is the same for all outputs that are connected to *x*. Thus, this fault will cause unidirectional error. □

4.2 Output Encoding

Another approach to prevent any two cubes from being *x*-bidirectional is output encoding. The idea behind output encoding is to ensure that the outputs corresponding to two input cubes C_i and C_j with dist(C_i, C_j) = 1, are not partially bidirectional.

To illustrate, let us consider Table I, where Z_1 and Z_2 are the symbolic outputs. In this Table, C_1 and C_3 are *a*-bidirectional. Thus, by changing the output encoding for Z_1 from 100 to 110, it can be guaranteed that the fault on line *a* will not produce bidirectional error at the output.

Let us assume the specification of a logic circuit with N symbolic outputs. An algorithm for encoding the output so that there are no *m*-bidirectional input cubes is given below.

Algorithm 3:

1. Initialize m to [lg *N*] where N is the number of symbolic outputs.
2. Construct an undirected graph G having the output symbols as vertices.
3. Connect Vertex O_i to vertex O_j if they are adjacent.
4. Construct a unidirectional graph G_m with 2^m nodes.
5. If Gm does not cover G, goto 6; else, an output encoding assignment is obtained from G_m. Exit.
6. Increment m. Goto step 4.

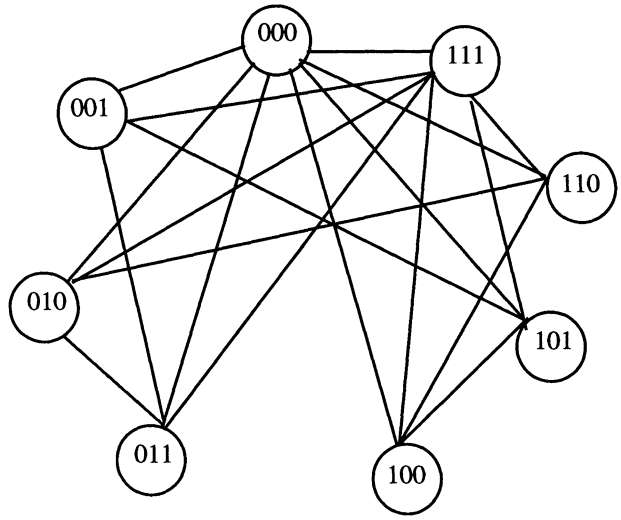

FIGURE 3. Unidirectional graph G_3 for m = 3.

Figure 3 shows the unidirectional graph G_3. Vertices which are represented by all 0's or all 1's are connected to all other vertices because there is no other vertex which is partially bidirectional with these vertices. The number of edges for a vertex depends on the number of 1's in the representation of that vertex. If k is the number of 1's in an m-bit vertex, then the number of edges for that vertex is $(2^k - 1) + (2^{(m-k)} - 1)$.

Table V shows that the number of edges each node has in a unidirectional graph as a function of m and k. For every (m, k) entry, the number of vertices is m!/(k! × (m-k)!). A fully connected subgraph of G_m can have a maximum of m + 1 nodes. For example, a fully connected subgraph of G_3 can have a maximum of 4 nodes.

To illustrate the application of the above algorithm, let us consider the specification for 3-bit priority encoder shown in Table VI. Each vertex in Graph G, Figure 4, is connected to all other vertices. The unidirectional graph G_2 does not cover G. However,

TABLE V
Adjacency for Each Node as a Function of Space Dimension (m) and number of 1's (k)

	k = 0	k = 1	2	3	4	5	6	7	8	9
m = 1	1	1								
2	3	2	3							
3	7	4	4	7						
4	15	8	6	8	15					
5	31	16	10	10	16	31				
6	63	32	18	14	18	32	63			
7	127	64	34	22	22	34	64	127		
8	255	128	66	38	30	38	66	128	255	
9	511	256	130	70	46	46	70	130	256	511

TABLE VI
Specification of 3-bit Priority Encoder

Input Cubes	Inputs x	y	z	Output s
C1	0	0	0	O1
C2	1	—	—	O2
C3	0	1	—	O3
C4	0	0	1	O4

G_3 covers G (the graph in Figure 3 covers the graph in Figure 4); hence, three bits are needed for output encoding.

Since G is fully connected, a fully connected subgraph containing 4 vertices of G_3 is needed for output encoding. One possible choice is (000, 111, 101, 100). Therefore, the outputs can be encoded as follows: $000 \rightarrow O_1$, $111 \rightarrow O_2$, $101 \rightarrow O_3$, $100 \rightarrow O_4$.

Step 5 of Algorithm 3 is a graph embedding problem which is NP complete. Therefore solving this embedding problem by using the smallest m requires exhaustive search. A heuristic solution with polynomial complexity is proposed in this paper. This heuristic solution gives satisfactory results when applied to MCNC benchmark circuits. As the results show, in the worst case the number of bits used for encoding the output is only one bit more than the minimum number required. The nodes of graph G are the symbolic outputs which are denoted by O_i's. The nodes of graph G_m are m-bit vectors which are denoted by V_i's. The graph-embedding procedure is performed as follows:

```
graph_embedding(G, Gm)
{
  G'' = G;
  G''m = Gm;
  repeat
  {
    G' = G'';
    G'm = G''m;
    Oi = get_max(G');
    unidirectional_set(Oi) = set of all nodes that are
      connected to Oi;
    Vi = get_max
    unidirectional_set(Vi) = set of all nodes that are
    connected to Vi;
    if num_edges(Oi) ≤ num_edges(Vi)
    {
      use the m-bit encoding for Vi to encode output
      Oi;
      form G'' which is a subgraph of G' with ver-
      tices as unidirectional_set(Oi);
      form G''m which is a subgraph of G'm with ver-
      tices as unidirectional_set(Vi);
      graph_embedding(G'', G''m);
      G'' = G' - (node Oi and unidirectional_set(Oi)
      and all edges joining the edges of these
      nodes);
      G''m = G'm - (node Vi and unidirectional_
      set(Vi) and all edges joining the edges of these
      nodes)
    }
    else
    {
      m = m + 1;
      undecode all symbolic outputs;
      G'' = G;
      G''m = Gm;
  } until all outputs are encoded;
}.
```

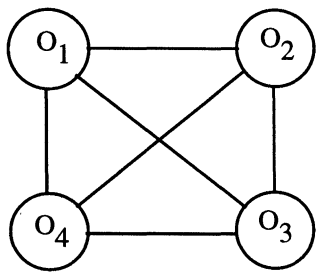

FIGURE 4. Graph G for 3-bit priority encoder.

The routine **get_max**( ) returns the node that has the largest number of edges. The routine **num_edges**( ) returns the number of edges for the node. **Unidirectional_set**(O_i) is the set of nodes that are joined to O_i by an edge. The dimension of G_m will increase by 1 when the **if** condition in **graph_embedding** ( ) fails; in other words, when node O_i in G' could not be mapped to node V_i in G'_m because O_i has more edges than V_i. When the **if** condition is true in the procedure, **graph_embedding**( ) will be recursively called for embedding smaller sizes graphs because the cardinality of the **unidirectional_set**(O_i) is always less than the number of nodes in the graph containing O_i.

Lemma 4: The complexity of Algorithm 3 does not exceed the order of N^4 where N is the number of symbolic outputs.

Proof: The number of edges/nodes in graph G are used in the estimation of the complexity because the number of edges/nodes in G_m are equal to a constant time the number of edges/nodes in G. The complexity of constructing graph G in Algorithm 3 is of the order of N^2 because for each node in G, N -1

checks are made to see if that node is connected to the other nodes (Step 3 in the algorithm). Similarly, the construction of G_m has polynomial complexity. Each call to the **graph_embedding( )** procedure is linearly related to the number of nodes plus the number of edges because routine **get_max( )** is linearly related to the number of nodes (N), and routine **num_edges( )** takes constant time. The **unidirectional_set ( )** is constructed in a time linearly related to the number of edges (|E|). It should be noted here that the complexity of the above procedures depends on how the graphs have been stored. When the **if** condition is always true in the procedure, **graph_embedding( )** will be recursively called for a maximum of N -1 times with each call requiring steps of order N. Thus, the complexity is of the order of N(N + |E|). The worst case occurs when G is fully connected; thus $|E| = N^2/2$ and the complexity of step 5 will be of the order of N^3. When the **if** condition fails in the last recursive call for **graph_embedding( )**, order of N^3 operations are wasted and m is increased by 1 and the whole procedure is repeated. The **if** condition will be true when m is equal to N − 1 because a fully connected subgraph from G_m of N nodes can be found. Consequently G will be covered by G_m. Therefore, the **if** condition can fail up to (N − 1 − [lg *N*]) times, and for each iteration, operations of the order of N^3 are needed, thus the worst case complexity is order of N^4. Q.E.D. □

Let us consider the application of the graph embedding procedure to the 3-bit priority encoder shown in Table VI. The graph G for the 3-bit priority encoder is shown in Figure 4. The procedure will try first **graph_embedding**(G, G_2). In the first iteration, 00 will be used for O_1 encoding. Thus, **unidirectional_set**(O_1) = {O_2, O_3, O_4}, and **unidirectional_set**(00) = {11, 01, 10}. Next, 11 will be used for O_2 encoding. Thus, **unidirectional_set**(O_2) = {O_3, O_4}, and **unidirectional_set**(11) = {01, 10}. In this case, O_3 could not be mapped to either 01 or 10 because num_edges(O_3) (= 1) is greater than **num_edges**(01) (= 0). Thus, the dimension is increased from 2 to 3, and **graph_embedding**(G, G_3) is called. It could be easily seen how 000 → O_1, 111 → O_2, 101 → O_3, 100 → O_4.

Lemma 5: If a prime and irredundant combinational logic circuit has its outputs encoded using algorithm 2, then any stuck-at fault (internal or at the input lines) will result in either single bit error or unidirectional multibit error at the output regardless of the way the circuit is implemented.

Proof: Algorithm 2 guarantees the elimination of *m*-bidirectional input cubes; therefore, by lemma 2, all single faults at the input lines will cause unidirectional error at the output. On the other hand, since the circuit is prime and irredundant, it has to be 100% testable. Therefore, for each node, there exists at least one input pattern that makes a sensitizable path from an input(s) through that node to an output(s). If there is only one sensitizable path for any input pattern from an input passing through node *x* to an output (there might be another path, but it cannot be sensitizable for the same input pattern), a fault at node *x* will produce single bit error since the fault only propagates to one output. In addition, if there is k sensitizable paths from input *a* through node *x* to k different outputs (Figure 2), an event at input *a* will cause an event at node *x* which will also cause events at the k outputs. An event at input *a* cannot cause a bidirectional change at the output due to the output assignment in algorithm 2. Thus, a transition or a stuck-at fault at node *x* cannot cause bidirectional error under the given input. Moreover, there might be other paths from an input different from *a* through node *x* and to some outputs; however, similar arguments can be made to prove that a fault at node *x* cannot cause bidirectional error. Consequently, all possible single stuck-at fault will either cause single bit error or unidirectional multibit error at the output. Q.E.D. □

The following lemma summarizes the main features of the proposed techniques.

Lemma 6: If a combinational circuit is designed such that all faults at the inputs which create bidirectional error at the output are removed, any fault in the circuit, internal or at the inputs, will result in either single bit error or unidirectional multibit error at the output irrespective of the way the circuit is implemented.

Proof: The proof of this lemma is similar to the proof of lemma 5. □

A detailed example is considered below to illustrate the application of the proposed techniques.

Example: In this example the application of the encoding techniques on **rd53** MCNC benchmark circuit will be considered. Table VIIa shows the description of the benchmark circuit. **rd53** circuit has 5 inputs, 3 outputs, 32 product terms, 32 different input patterns and 6 different output patterns. An

TABLE VII
PLA Descriptions of the Same Circuit Using Different Input and Output Encoding Schemes

(a) Benchmark description

.i 5	
.o 3	
.p 32	
1-111	1~~
11-11	1~~
1111-	1~~
111-1	1~~
-1111	1~~
01-01	~~1
-0110	~~1
001-1	~~1
1-001	~~1
1-100	~~1
110-0	~~1
011-0	~~1
1001-	~~1
0-011	~~1
-1010	~~1
-0101	~~1
01110	~1~
00010	~1~
01000	~1~
11111	~1~
00100	~1~
00111	~1~
11100	~1~
11010	~1~
01101	~1~
01011	~1~
10110	~1~
10000	~1~
11001	~1~
00001	~1~
10101	~1~
10011	~1~

(b) Input encoding.

.i 7	
.o 3	
.p 32	
111----	1~~
11-1---	1~~
11--1--	1~~
11---1-	1~~
11----1	1~~
1-11---	~~1
1-1-1--	~~1
1-1--1-	~~1
1-1---1	~~1
1--11--	~~1
1--1-1-	~~1
1--1--1	~~1
1---11-	~~1
1---1-1	~~1
1----11	~~1
-111---	~~1
-11-1--	~1~
-11--1-	~1~
-11---1	~1~
-1-11--	~1~
-1-1-1-	~1~
-1-1--1	~1~
-1--11-	~1~
-1--1-1	~1~
-1---11	~1~
--111--	~1~
--11-1-	~1~
--11--1	~1~
--1-11-	~1~
--1-1-1	~1~
--1--11	~1~
---111-	~1~

(c) Symbolic outputs

00000	O0
00001	O1
00010	O1
00011	O2
00100	O1
00101	O2
00110	O2
00111	O3
01000	O1
01001	O2
01010	O2
01011	O3
01100	O2
01101	O3
01110	O3
01111	O4
10000	O1
10001	O2
10010	O2
10011	O3
10100	O2
10101	O3
10110	O3
10111	O4
11000	O2
11001	O3
11010	O3
11011	O4
11100	O3
11101	O4
11110	O4
11111	O5

(d) Output encoding.

.i 5	
.o 3	
.p 32	
00000	111
00001	110
00010	110
00011	010
00100	110
00101	010
00110	010
00111	011
01000	110
01001	010
01010	010
01011	011
01100	010
01101	011
01110	011
01111	001
10000	110
10001	010
10010	010
10011	011
10100	010
10101	011
10110	011
10111	001
11000	010
11001	011
11010	011
11011	001
11100	011
11101	001
11110	001
11111	000

unconstrained boolean minimization using MIS [10] is applied first to the original description of the circuit as described in the benchmark, which produces the following results:

lits(sop) = 45 lits(fac) = 36, where lits(sop) is the number of literals in sum-of-product form, and lits(fac) is the number of literals in the factored form.

The output equations are:

$$\{v5.0\} = (5)\ v0\ v1 + v0\ v2\ v3\ v4 + v1\ v2\ v3\ v4$$

$$\{v5.1\} = (11)\ v0 + (11)'\ v0'$$

$$\{v5.2\} = \{v5.0\}'\ \{v5.1\}'\ v1 + \{v5.0\}'\ (5) + \{v5.0\}'\ v0\ v2 + \{v5.0\}'\ v0\ v3 + \{v5.0\}'\ v0\ v4$$

(4) = (14) v1 + (14)′ v1′

(5) = (14) v4 + (14)′ v2

(11) = (4) v4′ + (4)′ v4

(14) = v2 v3′ + v2′ v3

v0, v1, v2, v3, and v4 are the inputs; {v5.0}, {v5.1} and {v5.2} are the outputs; (4), (5), (11) and (14) are internal nodes. Each node is written in a sum-of-product form in terms of the input variables and the other nodes. In this realization, there exist some faults that create bidirectional error at the outputs. If, for instance, the input pattern is v0v1v2v3v4 = 11001, the internal nodes and the outputs will be as follows: (14) = (4) = (5) = 0; (11) = 1; {v5.0} = 0; {v5.1} = {v5.2} = 1. If, however, input v2 is s-a-1, the value of the nodes will be changed to: (14) = (4) = (5) = 0; (11) = 0; (11) = 0; {v5.0} = 1; {v5.1} = {v5.2} = 0. Thus, in the presence of the fault, the outputs change from 011 to 100. Similarly, a stuck-at-1 at node (14) will create bidirectional error when 11001 is applied at the input.

The standard cell representation of the circuit results in an area of 110400 Lambdas (230 × 480 Lambdas).

As mentioned previously, m-out-of-n codes are used for input encoding. To minimize the number of input lines, we choose m to be $[n/2]$. Since the number of input combinations is 32, 3-out-of-7 code is used for encoding the inputs as shown in Table VIIb. It can be easily verified that by using this input encoding, either single bit error or unidirectional multibit error at the output will result in presence of a stuck-at fault. An unconstrained boolean minimization using MIS is applied, and the following results are obtained:

lits(sop) = 41 lits(fac) = 32

The output equations are:

{v7.0} = (20) v0 v1 + v0 v1 v2 + v0 v1 v3

{v7.1} = (19) v1 + (19) v2 + (20) v1 v2 + v3 v4 v5

{v7.2} = (19) v0 + (20) v0 v2 + v0 v2 v3 + v1 v2 v3

(19) = (20) v3 + v4 v5 + v4 v6 + v5 v6

(20) = v4 + v5 + v6

In these output equations, no variable appear in both complemented and uncomplemented form. Thus, these equations are monotonic which guarantees that every fault will create either single bit error or unidirectional multibit error at the output (Lemma 3).

The standard cell layout for this realization results in an area of 109440 Lambdas (235 × 384 Lambdas). Thus, the overhead is (109440 − 110400)/110400 = −0.87%; i.e., the area is actually reduced.

Table VIIc shows the description of **rd53** when the inputs cubes are expanded and a symbolic output is assigned to each output pattern. The graph constructed from Algorithm 3 is shown in Figure 5.

Since G_3 covers G, a possible output encoding is as follows:

111 ⟶ O0 110 ⟶ O1 010 ⟶ O2

011 ⟶ O3 001 ⟶ O4 000 ⟶ O5

We applied an unconstrained boolean minimization using MIS, and the following results are obtained:

lits(sop) = 40 lits(fac) = 36

The output equations are:

{v5.0} = (32) (39) + (33) (38)′ v4′

{v5.1} = (32) + (33) (37) + (33) v4′ + (39)

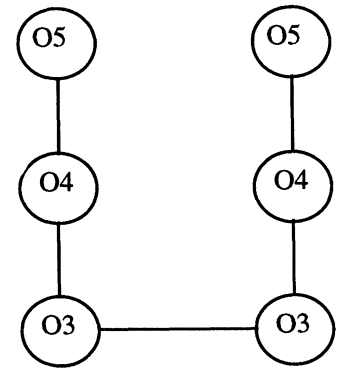

FIGURE 5. Graph G for the circuit in Table VIIc.

$$\{v5.2\} = (32)'(33)\ (39)' + (33)'\ v2\ v4' + (33)'\ v2'\ v3 + (33)'\ v3'\ v4 + (37)'\ (39)\ v4 + (38)'\ (39)\ v4'$$

$$(32) = (37)\ v4' + (38)'$$

$$(33) = v0' + v1'$$

$$(37) = v2' + v3'$$

$$(38) = v2 + v3$$

$$(39) = v0'\ v1'$$

Node (33) is used in an uncomplemented form in the equations for {v5.0}, {v5.1}, and {v5.2}; however, its is used in complemented form in {v5.2}. In this case, the num_inv((33), {v5.0}) = 0 whereas num_inv((33), {v5.2}) = 1. We will prove that if one of the paths from node (33) to an output where num_inv ((33), output} = 0 is sensitizable, all other paths from (33)′ to the outputs are not sensitizable. If the path from (33) to {v5.1} is sensitizable, then v4 should be 0 or (37) should be 1, and both (39) and (32) should be 0. When node (39) = 0, then

$$\{v5.2\} = (33) + (33)'v2v4' + (33)'v2'v3 + (33)'v3'v4$$
$$= (33) + v2v4' + v2'v3 + v3'v4.$$

(33) is only used in an uncomplemented form, and the path from (33)′ to {v5.2} is masked. Therefore, there will be the same number of inversions between node (33) and the outputs.

Similarly, it can be shown that if one of the paths from a node to an output, where the number of inversions is even(odd) is sensitizable, then all other paths from that node to the outputs where number of inversions is odd (even) are not sensitizable.

The standard cell layout for the above realization results in an area of 90688 Lambdas (218 × 416 Lambdas). Thus the area overhead is (90688 − 110400)/110400 = −18%.

Note that the area overhead for the standard cell layout is not the same as the overhead for lits(fac) (Tables IX and X). This is because the area in the number of literals does not take into account the routing needed in the layout. However, lits(fac) still give a good measure of the overhead of different realization.

5. RESULTS

The number of literals in the factored form (multilevel realization) is approximately twice the number of transistors used in CMOS implementation because each literal will be an input to two transistors (one n-type and the other is p-type). Therefore, the results reported here are the number of literals in both factored form and sum-of-product form. As far as we are aware, no other input/output encoding schemes are available that have the same objectives as that proposed in this paper. Thus, for each benchmark circuit, we will have three realizations: 1) direct implementation of the circuit as described in the benchmark, 2) implementation of the circuit after input encoding and 3) implementation of circuit after applying Algorithm 3. The results will be reported as the number of literals for each realization. The results of realization 2) and 3) are compared to the results of realization 1). The statistics of the benchmarks are given in Table VIII where #inp is the number of input lines, #out is the number of output lines, #pro is the number of product terms, #diff_i is the number of different inputs present in the pla description, and #diff_o is the number of different outputs after expanding the input cubes. An unconstrained boolean minimization has been applied to the different circuits by using the multilevel logic synthesis tool MIS [10].

Table IX shows the number of literals obtained for realizations 1) and 2). #lit(sop) is the number of literals in sum of product form, #lit(fac) is the number of products in the factored form.

The percentage overhead is given for literal counts obtained from both sum-of-product and factored representation. Negative overhead indicates that the encoded circuit has fewer literals than the original circuit. It is clear from the experimental results that one can apply the input encoding to ensure every fault result in either a single bit error or unidirectional multibit error at the output, without increasing and

TABLE VIII
Statistics of Benchmark Examples

Example	#inp	#out	#pro	#diff_i	#diff_o
5xp1	7	10	75	70	128
apex1	45	45	206	205	
bw	5	28	87	76	
clip	9	5	167	166	
con1	7	2	9	9	4
duke2	22	29	87	87	
misex1	8	7	32	18	
misex2	25	18	29	29	
rd53	5	3	32	32	6
rd73	7	3	141	141	8
table3	14	14	175	175	

TABLE IX
Literal Count for Original and Input Encoded Circuits Plus the Overhead

	Original description		Input Encoding (m-out-of-n)			ovd(%)	
Example	#lit (sop)	#lit (fac)	#lit (sop)	#lit (fac)	#inp	sop	fac
5xp1	165	123	148	119	8	−10.3	−3.2
apex1	1822	1557	1315	1081	10	−27.8	−30.6
bw	173	160	228	200	9	31.7	25
clip	190	141	125	97	10	−34.2	−31.2
con1	23	19	13	11	5	−43.5	−42.1
duke2	532	433	440	379	9	−17.3	−12.5
misex2	113	103	74	66	7	−34.5	−35.9
rd53	45	36	41	32	7	−8.9	−11.1
rd73	108	80	111	82	10	2.78	2.5
table3	1019	838	740	567	10	−27.4	−32.3
total	4190	3490	3235	2634		−19.4	−24.5

in most cases decreasing the overhead. The output encoding algorithm was applied to a few benchmark circuits to get some estimate of the overhead. Table X shows the results of these benchmarks.

In Table X, #out is the number of output encoding bits used in the algorithm, ovd is the percentage overhead. As in input encoding, the average overhead for the benchmarks circuits is negative, which indicates that on average output encoded circuits require less overhead than the original circuit description. In addition, the number of bits used for encoding the output is only one bit more than the minimum number of bits required. Finally, the output encoding algorithm gives better results than the input encoding algorithm because the output algorithm assign unidirectional codes for two adjacent outputs; this assignment will make one output dominant resulting in fewer number of literals in the final circuit [11].

6. CONCLUSION

Input and output encoding techniques proposed in this paper guarantee that all stuck-at faults, internal or at the inputs, will cause either single bit error or unidirectional multibit error at the output. Previously published work in this area was restricted to AND/OR type circuits, and both inputs and their inversions were assumed fault-free. Thus, there are no schemes or algorithms for output or input encoding that have the same objectives as presented in this paper. The techniques proposed allow multilevel implementation as well as consider the possibility of faults at the input lines. The input encoding algorithm can be applied to arbitrary circuits because m-out-of-n code is used for input encoding, so the increase in the input lines is not significant. The worst case theoretical complexity of Algorithm 3 is N^4; however, the average experimental complexity in N^2 (the number of bits used for encoding the output is one bit more than the minimum number of bits required). Therefore, Algorithm 3 can be efficiently applied to a circuit description that has up to few hundreds symbolic outputs.

TABLE X
Literal Count and Overhead for Output Encoding Algorithm

Ex.	#out	#lit (sop)	#lit (fac)	ovd (sop)	ovd (fac)
5xp1	8	24	24	−84.6%	−80.5%
con1	3	31	25	34.7%	31.6%
rd53	3	40	36	−11.1%	0.0%
rd73	3	78	73	−27.8%	−8.7%
total		173	158	−49.2%	−38.8%

References

[1] Y. Savaria, N.C. Rumin, J. Hayes, and V. Agrawal, "Soft-error filtering: A solution to the reliability problem of future VLSI logic circuits," *IEEE Proc.*, vol. 74, no. 5, pp. 669–683, May 1986.

[2] M. Yen, W. Fuchs, and J. Abraham, "Designing for concurrent error detection in VLSI: Application to a microprogram control unit," *IEEE Journal of Solid-State Circuits*, vol. SC-22, pp. 595–605, August 1987.

[3] J. Smith and G. Metze, "The design of totally self-checking combinational circuits," in *Proc. Int. Symp. Fault-Tolerant Computing*, Los Angeles, CA, pp. 130–134, June 1977.

[4] M. Diaz, "Design of totally self-checking and fail safe sequential machines," in *Proc. Int. Symp. Fault-Tolerant Computing*, Urbana, IL, pp. 9–24, June 1974.

[5] J. Smith and P. Lam, "A theory of totally self-checking system design," *IEEE Trans. on Computers*, vol. C-32, pp. 831–844, September 1983.

[6] C. Frieman, "Optimal error detecting codes for completely asymmetric binary channels," *Information and Control*, vol. 5, pp. 64–71, March 1962.

[7] H. Dong, "Modified Berger codes for detection of unidirectional error," in *Proc. Int. Symp. Fault-Tolerant Comput.*, Santa Clara, CA, pp. 317–320, June 1982.

[8] S. Devadas, H.T. Ma, A.R. Newton, and A.L. Sangiovanni-Vincentelli, "Irredundant sequential machines via optimal logic synthesis," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 8–17, Jan. 1990.

[9] S. Devadas, and K. Keutzer, "A unified approach to the synthesis of fully testable sequential machines," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 39–50, Jan. 1991.

[10] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "MIS: A multiple-level logic optimization program," *IEEE Trans. on Computer-Aided Design*, vol. 7, pp. 1062–1081, Nov. 1987.

[11] S. Devadas, and A.R. Newton, "Exact Algorithms for output encoding, state assignment and four-level Boolean minimization," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 13–27, Jan. 1991.

[12] K. Keutzer, S. Malik, and A. Saldanha, "Is redundancy necessary to reduce delays?," *IEEE Trans. Computer Aided-Design*, vol. 10, pp. 427–435, April 1991.

[13] K. Bartlett, R.K. Brayton, G.D. Hachtel, R.M. Jacoby, C.R. Morrison, R.L. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "Multi-level logic minimization using implicit don't cares," *IEEE Trans. Computer-Aided Design,* vol. 7, pp. 723–740, June 1988.

[14] J.E. Smith and G. Metze, "Strongly fault secure logic networks," *IEEE Trans. on Computers,* vol. C27, pp. 491–499, June 1987.

[15] S. Pagey, S.D. Sherlekar and G. Venkatesh, "A Methodology for the design of SFS/SCD circuits for a class of unordered codes," *Journal of Electronic Testing: Theory and Application,* 2, pp. 261–277, 1991.

Biographies

FADI BUSABA is an assistant professor in the department of Electrical Engineering at North Carolina A&T State University. His research interests include fault-tolerant computing, self-checking and synthesis for testability. He holds a BE from American University of Beirut, an M.S.E.E. from North Carolina Agricultural and Technical State University, and a Ph.D. in Computer Engineering from North Carolina State University at Raleigh. He received MCNC Fellowship and an IEEE member.

PARAG K. LALA is a professor in the department of Electrical Engineering at North Carolina Agricultural and Technical State University. His research interests include test generation/testability, fault-tolerant computing, digital system design, and self-checking design. He authored *Fault-Tolerant and Fault-Testable Hardware Design* and *Digital System Design Using PLDs,* both published by Prentice-Hall, Inc. He holds an MSc in Electrical Engineering from King's College, London, and a PhD from the City University of London.